FANUC Robot series

R-30*i*A/R-30*i*A Mate CONTROLLER

KAREL Function

OPERATOR'S MANUAL

B-83144EN/01

- **Original Instructions**

Before using the Robot, be sure to read the "FANUC Robot Safety Manual (B-80687EN)" and understand the content.

The products in this manual are controlled based on Japan's "Foreign Exchange and Foreign Trade Law". The export from Japan may be subject to an export license by the government of Japan.
Further, re-export to another country may be subject to the license of the government of the country from where the product is re-exported. Furthermore, the product may also be controlled by re-export regulations of the United States government.
Should you wish to export or re-export these products, please contact FANUC for advice.

In this manual we have tried as much as possible to describe all the various matters.
However, we cannot describe all the matters which must not be done, or which cannot be done, because there are so many possibilities.
Therefore, matters which are not especially described as possible in this manual should be regarded as "impossible".

SAFETY

1 SAFETY PRECAUTIONS

For the safety of the operator and the system, follow all safety precautions when operating a robot and its peripheral devices installed in a work cell.
In addition, refer to the "FANUC Robot SAFETY HANDBOOK (B-80687EN)".

1.1 WORKING PERSON

The personnel can be classified as follows.

Operator:
- Turns robot controller power ON/OFF
- Starts robot program from operator's panel

Programmer or teaching operator:
- Operates the robot
- Teaches robot inside the safety fence

Maintenance engineer:
- Operates the robot
- Teaches robot inside the safety fence
- Maintenance (adjustment, replacement)

- An operator cannot work inside the safety fence.
- A programmer, teaching operator, and maintenance engineer can work inside the safety fence. The working activities inside the safety fence include lifting, setting, teaching, adjusting, maintenance, etc..
- To work inside the fence, the person must be trained on proper robot operation.

During the operation, programming, and maintenance of your robotic system, the programmer, teaching operator, and maintenance engineer should take additional care of their safety by using the following safety precautions.

- Use adequate clothing or uniforms during system operation
- Wear safety shoes
- Use helmet

1.2 WORKING PERSON SAFETY

Working person safety is the primary safety consideration. Because it is very dangerous to enter the operating space of the robot during automatic operation, adequate safety precautions must be observed.
The following lists the general safety precautions. Careful consideration must be made to ensure working person safety.

(1) Have the robot system working persons attend the training courses held by FANUC.

FANUC provides various training courses. Contact our sales office for details.

(2) Even when the robot is stationary, it is possible that the robot is still in a ready to move state, and is waiting for a signal. In this state, the robot is regarded as still in motion. To ensure working person safety, provide the system with an alarm to indicate visually or aurally that the robot is in motion.
(3) Install a safety fence with a gate so that no working person can enter the work area without passing through the gate. Install an interlocking device, a safety plug, and so forth in the safety gate so that the robot is stopped as the safety gate is opened.

The controller is designed to receive this interlocking signal of the door switch. When the gate is opened and this signal received, the controller stops the robot in an emergency. For connection, see Fig.1.2 (a) and Fig.1.2 (b).

(4) Provide the peripheral devices with appropriate grounding (Class A, Class B, Class C, and Class D).
(5) Try to install the peripheral devices outside the work area.
(6) Draw an outline on the floor, clearly indicating the range of the robot motion, including the tools such as a hand.
(7) Install a mat switch or photoelectric switch on the floor with an interlock to a visual or aural alarm that stops the robot when a working person enters the work area.
(8) If necessary, install a safety lock so that no one except the working person in charge can turn on the power of the robot.

The circuit breaker installed in the controller is designed to disable anyone from turning it on when it is locked with a padlock.

(9) When adjusting each peripheral device independently, be sure to turn off the power of the robot.

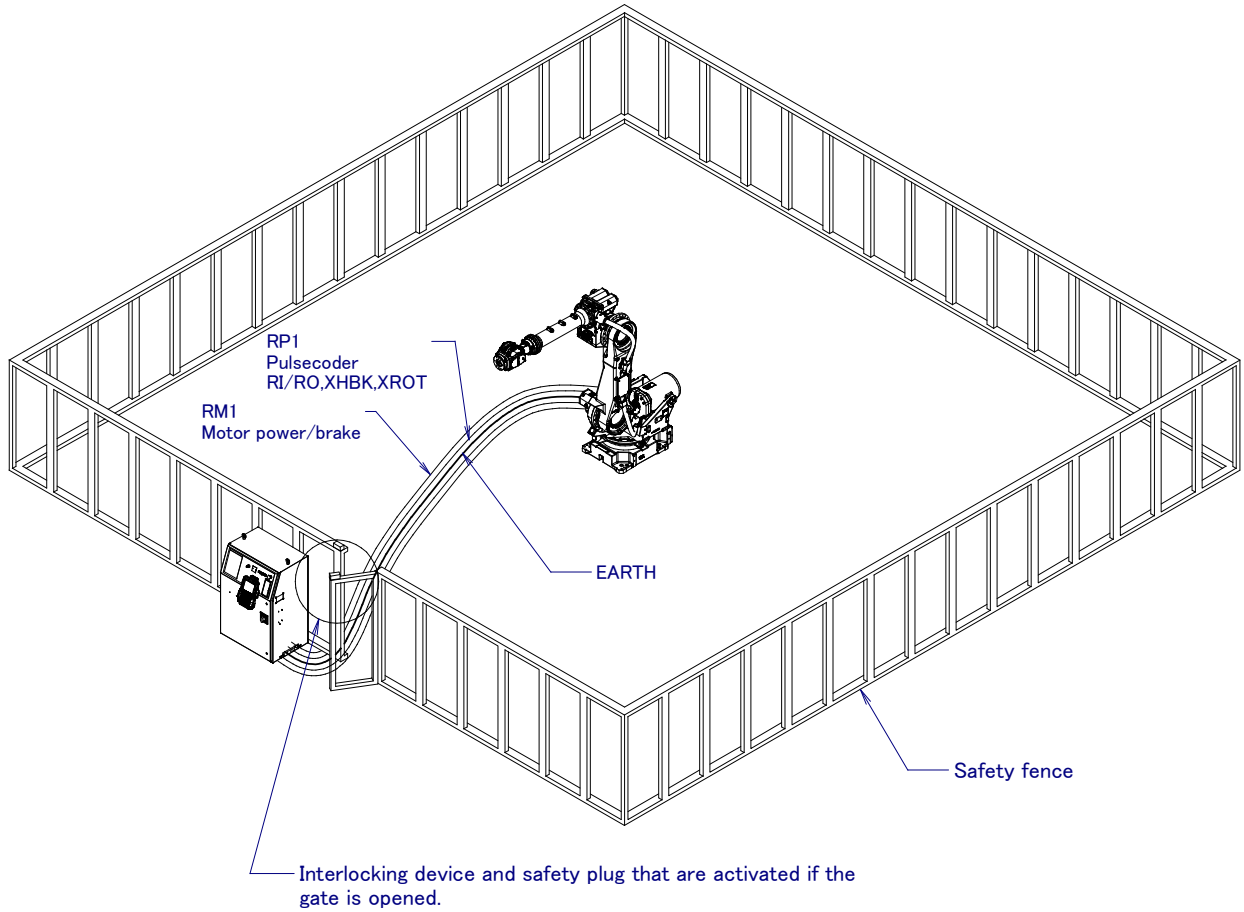


Fig. 1.2 (a) Safety Fence and Safety Gate

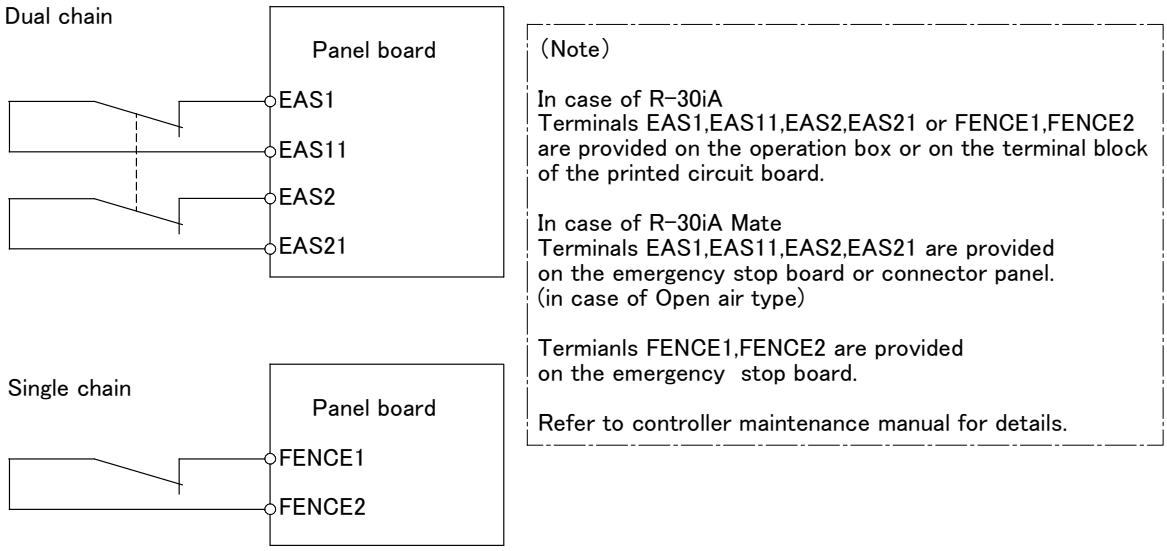


Fig.1.2 (b) Connection Diagram for Safety Fence

1.2.1 Operator Safety

The operator is a person who operates the robot system. In this sense, a worker who operates the teach pendant is also an operator. However, this section does not apply to teach pendant operators.

(1) If you do not have to operate the robot, turn off the power of the robot controller or press the EMERGENCY STOP button, and then proceed with necessary work.
(2) Operate the robot system at a location outside of the safety fence
(3) Install a safety fence with a safety gate to prevent any worker other than the operator from entering the work area unexpectedly and to prevent the worker from entering a dangerous area.
(4) Install an EMERGENCY STOP button within the operator's reach.

The robot controller is designed to be connected to an external EMERGENCY STOP button. With this connection, the controller stops the robot operation when the external EMERGENCY STOP button is pressed. See the diagram below for connection.

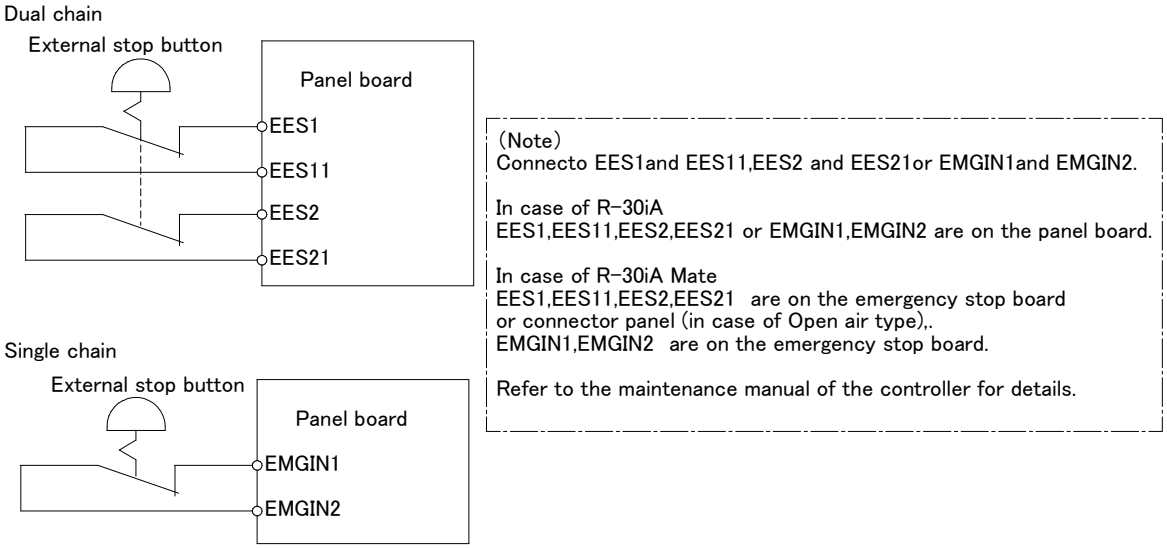


Fig.1.2.1 Connection Diagram for External Emergency Stop Button

1.2.2 Safety of the Teach Pendant Operator

While teaching the robot, the operator must enter the work area of the robot. The operator must ensure the safety of the teach pendant operator especially.

(1) Unless it is specifically necessary to enter the robot work area, carry out all tasks outside the area.
(2) Before teaching the robot, check that the robot and its peripheral devices are all in the normal operating condition.
(3) If it is inevitable to enter the robot work area to teach the robot, check the locations, settings, and other conditions of the safety devices (such as the EMERGENCY STOP button, the DEADMAN switch on the teach pendant) before entering the area.
(4) The programmer must be extremely careful not to let anyone else enter the robot work area.

Our operator panel is provided with an emergency stop button and a key switch (mode switch) for selecting the automatic operation mode (AUTO) and the teach modes (T1 and T2). Before entering the inside of the safety fence for the purpose of teaching, set the switch to a teach mode, remove the key from the mode switch to prevent other people from changing the operation mode carelessly, then open the safety gate. If the safety gate is opened with the automatic operation mode set, the robot enters the emergency stop state. After the switch is set to a teach mode, the safety gate is disabled. The programmer should understand that the safety gate is disabled and is responsible for keeping other people from entering the inside of the safety fence. (In case of R-30*i*A Mate Controller standard specification, there is no mode switch. The automatic operation mode and the teach mode is selected by teach pendant enable switch.)

Our teach pendant is provided with a DEADMAN switch as well as an emergency stop button. These button and switch function as follows:
(1) Emergency stop button: Causes an emergency stop when pressed.
(2) DEADMAN switch: Functions differently depending on the mode switch setting status.
  (a) Automatic operation mode: The DEADMAN switch is disabled.
  (b) Teach mode: Causes an emergency stop when the operator releases the DEADMAN switch or when the operator presses the switch strongly.

Note)The DEADMAN switch is provided to place the robot in the emergency stop state when the operator releases the teach pendant or presses the pendant strongly in case of emergency. The R-30*i*A/ R-30*i*A Mate employs a 3-position DEADMAN switch, which allows the robot to operate when the 3-position DEADMAN switch is pressed to its intermediate point. When the operator releases the DEADMAN switch or presses the switch strongly, the robot enters the emergency stop state.

The operator's intention of starting teaching is determined by the controller through the dual operation of setting the teach pendant enable/disable switch to the enable position and pressing the DEADMAN switch. The operator should make sure that the robot could operate in such conditions and be responsible in carrying out tasks safely.

The teach pendant, operator panel, and peripheral device interface send each robot start signal. However the validity of each signal changes as follows depending on the mode switch and the DEADMAN switch of the operator panel, the teach pendant enable switch and the remote condition on the software.

In case of R-30*i*A Controller or CE or RIA specification of R-30*i*A Mate Controller

Mode	Teach pendant enable switch	Software remote condition	Teach pendant	Operator panel	Peripheral device
AUTO mode	On	Local	Not allowed	Not allowed	Not allowed
		Remote	Not allowed	Not allowed	Not allowed
	Off	Local	Not allowed	Allowed to start	Not allowed
		Remote	Not allowed	Not allowed	Allowed to start
T1, T2 mode	On	Local	Allowed to start	Not allowed	Not allowed
		Remote	Allowed to start	Not allowed	Not allowed
	Off	Local	Not allowed	Not allowed	Not allowed
		Remote	Not allowed	Not allowed	Not allowed

In case of standard specification of R-30*i*A Mate Controller

Teach pendant enable switch	Software remote condition	Teach pendant	Peripheral device
On	Ignored	Allowed to start	Not allowed
Off	Local	Not allowed	Not allowed
	Remote	Not allowed	Allowed to start

(5) (Only when R-30*i*A Controller or CE or RIA specification of R-30*i*A Mate controller is selected.) To start the system using the operator's panel, make certain that nobody is the robot work area and that there are no abnormal conditions in the robot work area.
(6) When a program is completed, be sure to carry out a test run according to the procedure below.
  (a) Run the program for at least one operation cycle in the single step mode at low speed.
  (b) Run the program for at least one operation cycle in the continuous operation mode at low speed.
  (c) Run the program for one operation cycle in the continuous operation mode at the intermediate speed and check that no abnormalities occur due to a delay in timing.
  (d) Run the program for one operation cycle in the continuous operation mode at the normal operating speed and check that the system operates automatically without trouble.
  (e) After checking the completeness of the program through the test run above, execute it in the automatic operation mode.
(7) While operating the system in the automatic operation mode, the teach pendant operator should leave the robot work area.

1.2.3 Safety of the Maintenance Engineer

For the safety of maintenance engineer personnel, pay utmost attention to the following.

(1) During operation, never enter the robot work area.
(2) Except when specifically necessary, turn off the power of the controller while carrying out maintenance. Lock the power switch, if necessary, so that no other person can turn it on.
(3) If it becomes necessary to enter the robot operation range while the power is on, press the emergency stop button on the operator panel, or the teach pendant before entering the range. The maintenance personnel must indicate that maintenance work is in progress and be careful not to allow other people to operate the robot carelessly.
(4) When disconnecting the pneumatic system, be sure to reduce the supply pressure.
(5) Before the start of teaching, check that the robot and its peripheral devices are all in the normal operating condition.
(6) Do not operate the robot in the automatic mode while anybody is in the robot work area.
(7) When you maintain the robot alongside a wall or instrument, or when multiple workers are working nearby, make certain that their escape path is not obstructed.
(8) When a tool is mounted on the robot, or when any moving device other than the robot is installed, such as belt conveyor, pay careful attention to its motion.

(9) If necessary, have a worker who is familiar with the robot system stand beside the operator panel and observe the work being performed. If any danger arises, the worker should be ready to press the EMERGENCY STOP button at any time.
(10) When replacing or reinstalling components, take care to prevent foreign matter from entering the system.
(11) When handling each unit or printed circuit board in the controller during inspection, turn off the circuit breaker to protect against electric shock.
If there are two cabinets, turn off the both circuit breaker.
(12) When replacing parts, be sure to use those specified by FANUC.
In particular, never use fuses or other parts of non-specified ratings. They may cause a fire or result in damage to the components in the controller.
(13) When restarting the robot system after completing maintenance work, make sure in advance that there is no person in the work area and that the robot and the peripheral devices are not abnormal.

1.3 SAFETY OF THE TOOLS AND PERIPHERAL DEVICES

1.3.1 Precautions in Programming

(1) Use a limit switch or other sensor to detect a dangerous condition and, if necessary, design the program to stop the robot when the sensor signal is received.
(2) Design the program to stop the robot when an abnormal condition occurs in any other robots or peripheral devices, even though the robot itself is normal.
(3) For a system in which the robot and its peripheral devices are in synchronous motion, particular care must be taken in programming so that they do not interfere with each other.
(4) Provide a suitable interface between the robot and its peripheral devices so that the robot can detect the states of all devices in the system and can be stopped according to the states.

1.3.2 Precautions for Mechanism

(1) Keep the component cells of the robot system clean, and operate the robot in an environment free of grease, water, and dust.
(2) Employ a limit switch or mechanical stopper to limit the robot motion so that the robot or cable does not strike against its peripheral devices or tools.
(3) Observe the following precautions about the mechanical unit cables. When theses attentions are not kept, unexpected troubles might occur.
- Use mechanical unit cable that have required user interface.
- Don’t add user cable or hose to inside of mechanical unit.
- Please do not obstruct the movement of the mechanical unit cable when cables are added to outside of mechanical unit.
- In the case of the model that a cable is exposed, Please do not perform remodeling (Adding a protective cover and fix an outside cable more) obstructing the behavior of the outcrop of the cable.
- Please do not interfere with the other parts of mechanical unit when install equipments in the robot.

1.4 SAFETY OF THE ROBOT MECHANISM

1.4.1 Precautions in Operation

(1) When operating the robot in the jog mode, set it at an appropriate speed so that the operator can manage the robot in any eventuality.

(2) Before pressing the jog key, be sure you know in advance what motion the robot will perform in the jog mode.

1.4.2 Precautions in Programming

(1) When the work areas of robots overlap, make certain that the motions of the robots do not interfere with each other.
(2) Be sure to specify the predetermined work origin in a motion program for the robot and program the motion so that it starts from the origin and terminates at the origin.
Make it possible for the operator to easily distinguish at a glance that the robot motion has terminated.

1.4.3 Precautions for Mechanisms

(1) Keep the work areas of the robot clean, and operate the robot in an environment free of grease, water, and dust.

1.4.4 Procedure to move arm without drive power in emergency or abnormal situations

For emergency or abnormal situations (e.g. persons trapped in or by the robot), brake release unit can be used to move the robot axes without drive power.
Please refer to controller maintenance manual and mechanical unit operator's manual for using method of brake release unit and method of supporting robot.

1.5 SAFETY OF THE END EFFECTOR

1.5.1 Precautions in Programming

(1) To control the pneumatic, hydraulic and electric actuators, carefully consider the necessary time delay after issuing each control command up to actual motion and ensure safe control.
(2) Provide the end effector with a limit switch, and control the robot system by monitoring the state of the end effector.

TABLE OF CONTENTS

1 INTRODUCTION

This manual explains KAREL function integrated in R-30*i*A CONTROLLER containing FANUC Robot software (called the robot controller hereinafter). Please see each tool manual if you would like to know the way to operate the robot controller.
Content of this manual is only used in R-30*i*A robot controller.

This manual does not describe all function of KAREL. Please see following manual if you would like to know detail.
KAREL Reference Manual B-83144EN-1.

⚠CAUTION
Before the robot is started, it should be checked that no one is in the area of the safety fence. At the same time, a check must be made to ensure that there is no risk of hazardous situations. If detected, such a situation should be eliminated before the operation.

2 OVERVIEW

2.1 WHAT IS KAREL?

Karel Capek who is the Czechoslovakian author published the drama, R.U.R in 1920. Robot is described in it for the first time in the history. KAREL is come from his first name.

KAREL is the robot language for the robot system architecture. User's original function can be created after creating KAREL program on PC, loading it to the robot controller and executing it.

The program created with KAREL is able to execute on the robot controller. It is the same of the program of creating on teach pendant (called TP program hereinafter to distinguish KAREL program) in the point that it can be executed on the robot controller. Different point is the purpose of using. TP program is to execute robot motion and application instruction. KAREL program is to build up the robot system.

TP program can be created, edited, and executed on teach pendant, but KAREL program cannot be created and edited on the robot controller. Creating KAREL program on PC and converting it (called translate hereinafter), then execution form program is executed after loading it to the robot controller. TP program can be changed in necessity of usual operating, but KAREL program is created the time of building up the robot system and never to be changed in usual operating.

KAREL program that is loaded to the controller and set it to be executed automatically in the timing of power-on or showing some screen is recognized a part of system software of the robot system architecture for the robot system operator. Then it is no longer the usual program for him/her.

2.2 KAREL FEATURE

Listed KAREL feature is following.

- High-level language based on PASCAL.
  - Compile language
  - Allows using control structure of structured language such as repeating, selecting and so on.
  - Allows using calculation including transcendental function, comparing, logic and bit operation.
  - Allows using variables that is defined by user.
  - Allows using structure and array.
- Allows loading programs as the part of robot system.
- Allows using many function of the robot controller with various built-in function.
- Allows handling signals and variable event independent of program sequence.
- Allows operating input/outputting of file or screen.
- Allows sending or receiving the data from/to external by serial port or Ethernet.
  (It needs A05B-2500-R648 option if using socket message)

KAREL has functions of handling vector and position data and I/O events independent of the robot controller (called "condition handler" in KAREL hereinafter) adding PASCAL functions. Furthermore many built-in functions that control the robots or the robot controller are prepared as the standard, then users can create their own system.

TP program features controlling the robot motion sequence, KAREL program features controlling except the robot motion sequence. KAREL can realize the user or system original function not changing the robot controller system software.

KAREL program is used after creating on the PC and loading to the robot controller. Once loading it, you can use its function. Reloading is necessary as the same of TP program after init start.

2.3 FEASIBLE KAREL FUNCTION

Following is the feasible function using KAREL.

Creating Original screen

Original screen of teach pendant can be created by KAREL program. It can set system unique data. Created screen is allowed to use other standard screen by setting it to screen on the robot controller. It is not necessary to describe the data input and showing it on the screen, handling cursor and so in the KAREL. They are all controlled by system software screen control.

Simplified PLC function

KAREL program scans periodically the I/O input/output or some data and set them to some specified value if they are changed to specified value. This function is allowed to use simplified PLC function. In this function, it is allowed to handle system variables and register that is not used in integrated PMC.

Teaching robot position function

KAREL program allows setting a value the position register or TP program position data. It allows setting the calculated position data to the position register and current position in the timing of signal input to the position data of TP program. It allows you to create position teaching with external signal by combining manual-feed of the robot.

Numeric calculation function

TP program allows calculating only four operations of integer and real number. KAREL allows calculating transcendental function such as trigonometric function, logarithmic function, irrational function and so on, and matrix. Necessary calculating function that is calculated by KAREL is used with calling as sub program from TP program.

Data monitor and record

KAREL program is executed in the same timing of executing TP program that has motion group by using multi task. It allows monitoring the process and recoding it with executing KAREL periodically. Recorded data allows outputting external device such as memory card as the text file.

2.4 CAUTION OF USING KAREL

Please notice following caution when you use KAREL.

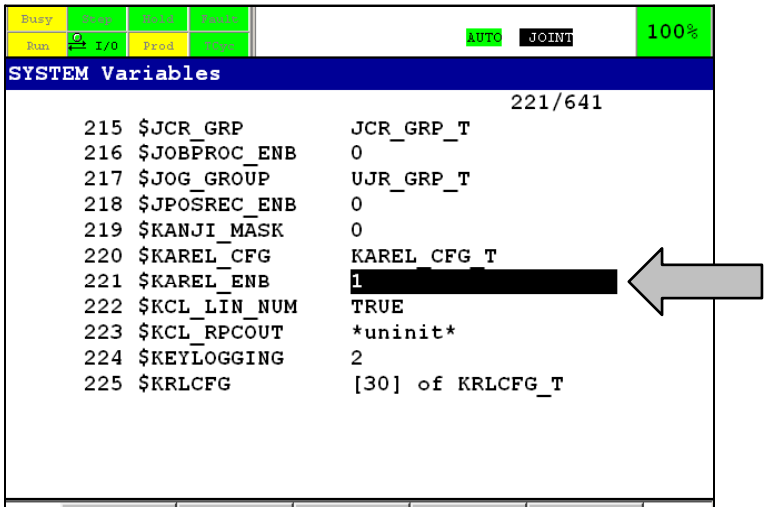
- KAREL program cannot control the robot working operation. You must use TP program when you would like to control the robot motion.
- The minimum frequency of scanning I/O or data concurrently running KAREL program is 8msec. It cannot scan under 8msec.
- It has limitation to scan I/O or data in the same timing of running KAREL program. And only limited process can be executed. If you would like to use full-dress PLC function, please use integrated PMC not using KAREL.
- In the case of repeating KAREL program in short period, system performance gets lower in compliance with increasing KAREL process. You must pay attention about this. It is realistically impossible to do complex process in short period such as 8msec.

- Actually data spread is occurred very much when some process executes periodically in KAREL program. Please use the condition handler.
- It takes a certain time to read or write the data from file or KAREL program. Therefore it is not allowed to input/output from/to the file in the case that input/output the file quickly.
- It is impossible to change existed function from KAREL program. For example it will be result in unexpected result if you change the system variable of existed function.
- It turns to system software version dependent program if system variable is referred directly. You must translate KAREL program in each software version and make the execution form program in each version if system variable is referred directly. Using GET_VAR and SET_VAR built-ins to refer system variable, it is system software independent KAREL program. Then once created execution form, it can use other robot controller. But it is not allowed using GET_VAR and SET_VAR in condition handler. And if you would like to handle quickly, these are not used.
- The power failure recovery is not applied to the KAREL program even if it is enable. After cycle power KAREL program is executed at the first line. You must determine the specification of KAREL thinking about this.

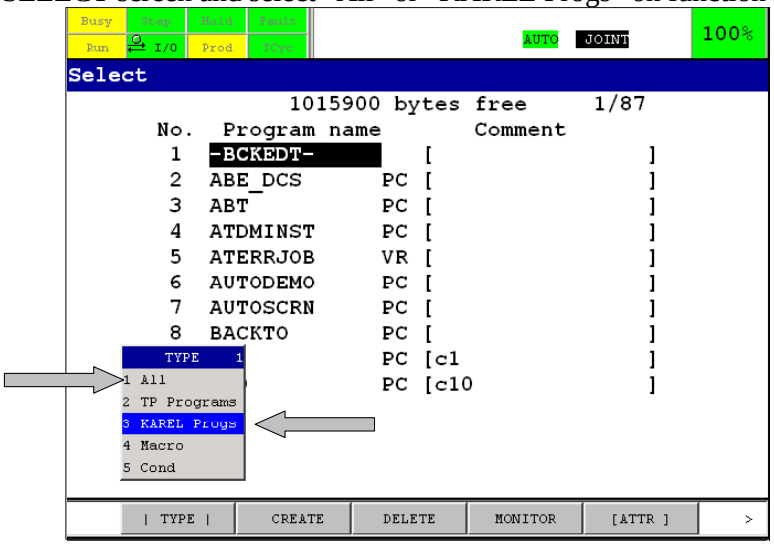
3 SYNTAX OF KAREL PROGRAM

Source code of KAREL program is a text file that has KL as the extension.
Execution form of KAREL program is created by translating the text file of KL extension. Execution form is the binary file that has the extension of PC.

For example translating AAA.KL gets AAA.PC.
AAA program is shown on the SELECT screen of teach pendant after loading AAA.PC to the robot controller. This AAA program can be selected and executed like TP program. Set the system variable $KAREL_ENB to 1 when you would like to show .PC file on SELECT screen.



Select F1[YTPE] in SELECT screen and select “All” or “KAREL Progs” on function key menu.

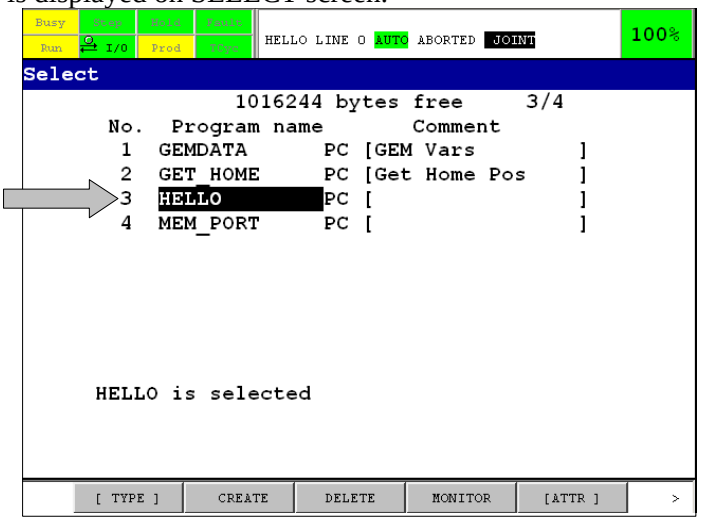


Showing a simple program as an example.

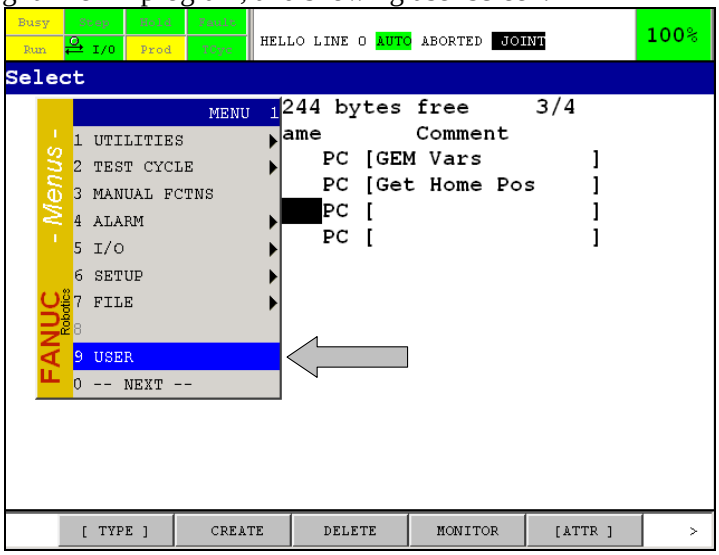
Following is the KAREL program sample to display “Hello, world” onto the user screen of teach pendant and to keep waiting for inputting 0 from the teach pendant. Detail of syntax is described later. It is called HELLO.KL.

```
******************************************************************************
PROGRAM hello
%NOPAUSE = ERROR+COMMAND+TPENABLE
VAR
   ent_val     : INTEGER
   exit_loop : BOOLEAN
BEGIN
   WRITE ( CR, CR, CR, CR, CR, CR, CR, CR, CR, CR )
   exit_loop = FALSE
   REPEAT
      WRITE ( 'Hello,world', CR )
      WRITE ( '0 END : ' )
      READ( ent_val )
      IF ent_val = 0 THEN
         exit_loop = TRUE
      ENDIF
   UNTIL exit_loop
   WRITE ( 'Done.', CR )
END hello
******************************************************************************
```

Translating HELLO.KL creates binary file, HELLO.PC.
Loading HELLO.PC, it is displayed on SELECT screen.



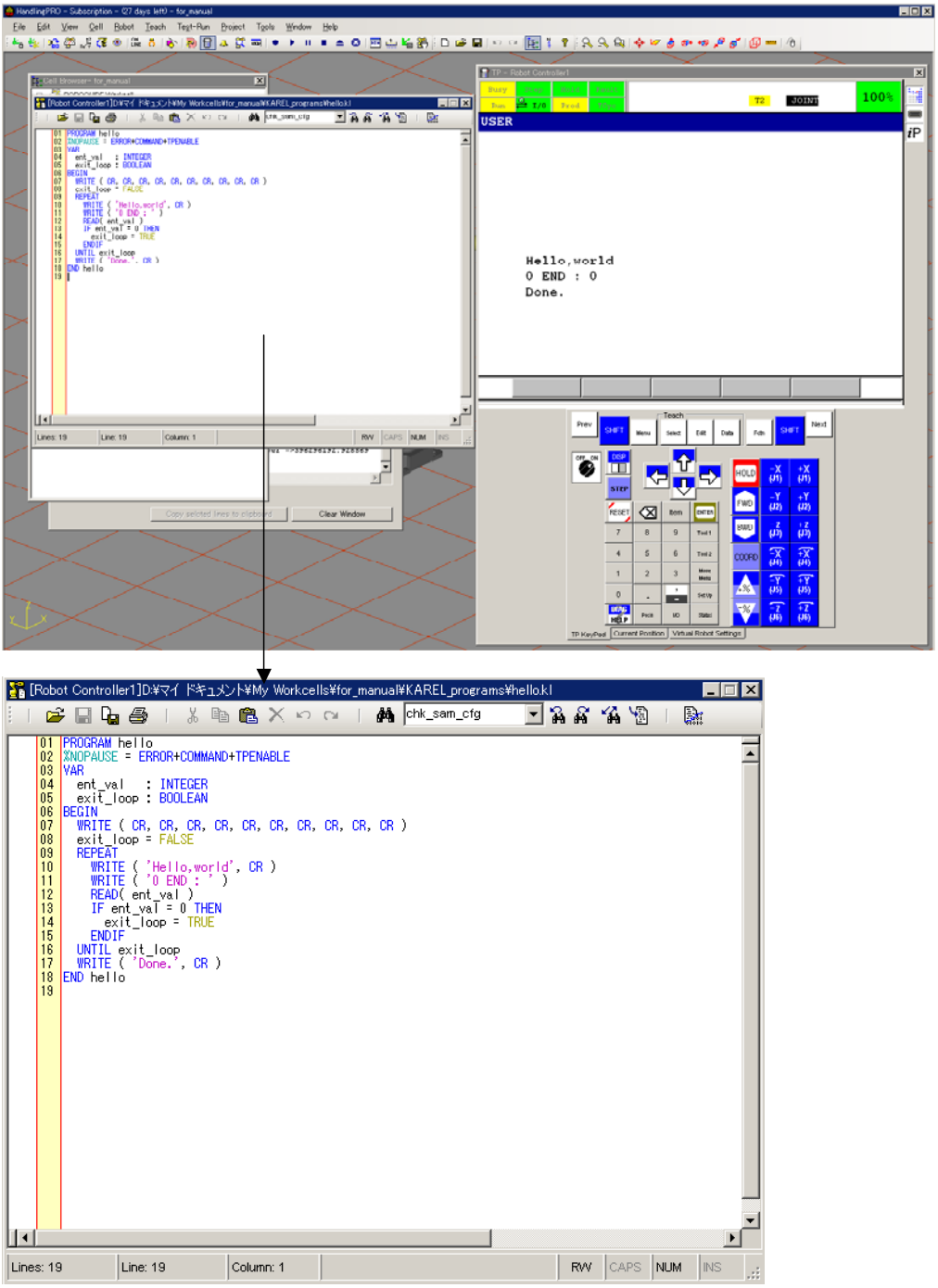
Selecting and executing it like TP program, and showing user screen.



Then you see following.

“Hello world” is kept showing until inputting 0 from teach pendant.

Above operation of program creating, translating it on PC is needed to create KAREL program. It is done with ROBOGUIDE, FANUC offline programming PC software. See chapter 4 to know detail about it.



Edit screen of KAREL source code

3.1 OVERVIEW OF THE ELEMENT OF KAREL PROGRAM

KAREL syntax is based on PASCAL. There is no concept to refer directly variable area with pointer. Following is the format of KAREL program source code (KL file).

PROGRAM program_name
% Program attribute
%ENVIRONMENT environment_file
%INCLUDE include_file

```
CONST
   Declaration of constant

TYPE
   Declaration of user type

VAR
   Declaration of variable

ROUTINE routin_name
   Describing of routine
END routin_name

BEGINE
   Describing of main routine
END program_name
```

In KAREL reserved word and built-in function are described uppercase letter, and others are described lower case letter. Following is concrete information of above. In addition, starting “--“ line in the KAREL source code is the comment.

```
PROGRAM my_test1
-- Specifying the program attribute. This line begins --, then it is the comment line.
%NOLOCKGROUP
%NOPAUSE=COMMAND+ERROR+TPENABLE

-- Specifying the environment file (environment file is the binary file, the extension is EV)
%ENVIRONMENT REGOPE

-- Specifying include file (include file is the text file, the extension is KL)
%INCLUDE kliotyps
%INCLUDE my_const

-- Declaration of the constant (declaration AAA_C and BBB_C)
CONST
AAA_C = 12
BBB_C = 14

-- Declaration of user type (definition the type of AAAAA_T)
TYPE
   AAAAA_T = STRUCTURE
      id1     : INTEGER
      name1 : STRING[12]
      id2     : INTEGER
      name2 : STRING[12]
   ENDSTRUCTURE
-- Declaration of variables
-- Declaration here is the global variables that enables in whole of this program.
-- And it allows referring from other programs.
VAR
aaaaa_var    : AAAAA_T
spot_count : INTEGER
num_of_gun : INTEGER
do_resume    : BOOLEAN
```

```
-- Declaration of external routine.
-- Declaration for using routine of other KL files in this KL file.
ROUTINE my_rtn2 FROM my_test2

-- Declaration of internal routine.
ROUTINE my_rtn1
VAR
   -- Declaration of variables
   -- Declaration here is the local variables that can only use in this routine.
BEGIN
   Describing local routine.
END my_rtn1

-- Beginning of the main program
BEGIN

-- Describing the main program.
-- Various instructions, built-in function, routine call, numeric calculation
IF move_flag THEN
aaaaa_var.id1 = 1
aaaaa_var.name1 = 'My test 1'
ELSE
aaaaa_var.id2 = 2
aaaaa_var.name2 = 'My test 2'
ENDIF

END my_test1
```

It has no need to describe specifying attribute, environment file, include file, constant, user type, variable and declaration routine if main program is not used. In the case of HELLO.KL, only main program is described because attribute is the default, no use of user type, variable and routine.

3.2 DETAIL OF THE ELEMENT OF KAREL PROGRAM

3.2.1 Usable character and symbol

Alphabet of upper or lower case letter, half size of kana, number and half size space are allowed.
Symbol @ < > = / * + - _ , ; : . # $ ' [ ] ( ) & % { } are allowed.

3.2.2 Rule of User-Defined

User determines the name of variables, constants, user type and so on. Following is its rule.

- Can have a maximum of 12 characters.
- Start with a letter.
- Cannot be reserved words.
- Must be defined before using.
- Can use upper or lower letter

We recommend following coding style.

Uppercase letter
- Constant.

- Defined type name and reserved words.
- Built-in function, program and routine name

Lowercase letter
- User-Defined type name
- Variables
- Comment

Above recommended coding style is to unify the description and to read easily. Not coding above style does not occur errors while translating.

3.2.3 Comments

A comment is marked by a pair of consecutive hyphens "--". On a program line, anything to the right of these hyphens is treated as a comment.

Comments are allowed describing English or Japanese. We recommend describing comments positively for getting KAREL program maintenance easier.

Comments are deleted automatically in the time of translating therefore it is no affection to the executing process of KAREL if you describe a lot of comments.

3.2.4 Specifying Attribute

It allows you to specify KAREL program attribute at timing of translating. Following is the explanation of spicing attribute.

%COMMENT = 'THIS IS COMMENT'

Specifies a comment of up to 16 characters. During load time, the comment is stored as a program attribute and can be displayed on the SELECT screen of the teach pendant.

%NOLOCKGROUP

WARNING
You must set this attribute in usual KAREL program.

Specifies that motion groups do not need to be locked when calling this program, or a routine defined in this program. Please set this attribute if the program for robot motion is executed. If it is not specified, the KAREL program is specified using all motion groups.

%SYSTEM

Specifies system program attribute to the KAREL program. Then the KAREL program is not counted as user program, it runs as the same time of TP program. System program is executed consecutively if it is single step mode.

%NOBUSYLAMP

Specified that the busy lamp will be OFF during execution. It allows external PLC to ignore the KAREL program of executing all time.

%NOPAUSE=ERROR+COMMAND+TPENABLE

Specifies a mask for pausing. This attribute is to ignore to pause if the event written after "=" is occurred.
The bits for the mask are as follows:
ERROR: ignore pause error severity
COMMAND: ignore pause command

TPENABLE: ignore paused request when TP enabled.
Specifying like above, once running, the program is not paused. However the program is paused if error is occurred in itself even if setting %NOPAUSE = ERROR.

%NOABORT=ERROR+COMMAND

Specifies the mask of aborting. This attribute is to ignore to abort if the event written after "=" is occurred.
ERROR: ignore abort error severity.
COMMAND: ignore abort command (CSTOPI input is ON means aborting from FCTN).
If setting like above, once program run, program is never aborted if any trouble occurs. However, when the error occurs in itself the program is aborted even if %NOABORT = ERROR was set.

%NOPAUSESHFT

Specifies that the task is not paused if SHIFT key is released. Normally the TP program run from the teach pendant is paused when the SHIFT key is released. But if this attribute is set, program is not paused after releasing of SHIFT key.

%TPMOTION

Specifies that task motion is enabled when the teach pendant is on.

%CMOSVARS

Specifies the default storage for KAREL variables is CMOS (permanent memory). Variables in CMOS is kept if cycle power. But CMOS memory size is very limited, then in some case of defining many variables, KAREL program is not loaded to the robot controller. Not setting this attribute, defined variables are in DRAM.

%STACKSIZE = n

Specifies stack size in long words. The default value of n is 300 (1200 bytes). It is needed to increase stack size in the case of stack overflow error occurs while executing the KAREL program.

3.2.5 Environment Files

Environment file specifies definitions for system variables and built-in function. The extension of it is EV of binary files. You must use it supplied by FANUC.

%ENVIRONMENT sysdef -- Declarations of system variables.
%ENVIRONMENT regope -- Declaration of built-in function associated with register.

Declaration of system variables is used in the only case of referred them directly. Please do not specify if built-in function, GET_VAR or SET_VAR is used. If sysdef environment is set, the KAREL program is system software dependent even though system variables are not referred directly.

Declaration file of built-in function is specified when using built-in function. Detail of built-in function is described in Appendix A ALPHABETICAL DESCRIPTION of KAREL Reference Manual. SYSTEM or PBCORE environment file is referred while translating, then these environment files are not necessary to describe.

3.2.6 Include Files

Include file requires the text file whose extension is KL. After specifying include file, it is loaded to specified position in the time of translating. For example, defining

```
CONST
   my_number = 1
```

into my_const.kl and specifying following,

```
PROGRAM test
%INCLUDE my_const
VAR
   Cur_number : INTEGER
BEGIN
   Cur_number = my_number
END test
```

means the same of following sample program.

```
PROGRAM test
CONST
   my_number = 1
VAR
   Cur_number : INTEGER
BEGIN
   Cur_number = my_number
END test
```

Include file is used for definition of common variables or constants among multi KAREL program or using user type in one file.

In addition to user-defined include file, there are supplied by FANUC files as follows.

%INCLUDE kliotyps -- I/O definition
%INCLUDE klevccdf -- Definition of character code of controlling screen
%INCLUDE klevkeys -- Definition of teach pendant key code.

These files are in following directory installed ROBOGUIDE.
C:¥Program Files¥FANUC¥WinOPLC¥Versions¥V730-1¥support (7DA3 series)
C:¥Program Files¥FANUC¥WinOPLC¥Versions¥V740-1¥support (7DA4 series)
C:¥Program Files¥FANUC¥WinOPLC¥Versions¥V750-1¥support (7DA5 series)
C:¥Program Files¥FANUC¥WinOPLC¥Versions¥V760-1¥support (7DA6 series)
C:¥Program Files¥FANUC¥WinOPLC¥Versions¥V770-1¥support (7DA7 series)

3.2.7 Defined Data Type

KAREL allows using the following data type.

Date Type	Detail
BOOLEAN	Represents the BOOLEAN predefined constants TRUE, FALSE, ON AND OFF, 4 bytes total.
INTEGER	4 bytes total.
REAL	32bit floating-point. 4 bytes total.
STRING	Data length is specified in variable definition up to 254. tmp_str : STRING[10]
VECTOR	Has following fields X : REAL Y : REAL Z : REAL

Date Type	Detail
POSITION	Consists of a matrix defining the normal, orient, approach and location vectors and a component specifying a configuration string, for a total of 56 bytes. NORMAL: VECTOR ORIENT: VECTOR APPROACH: VECTOR LOCATION: VECTOR CONFIG_DATA: CONFIG
XYZWPR	An XYZWPR consists of three REAL components specifying a Cartesian location (x, y, z), three REAL components specifying an orientation (w, p, r), and a component specifying a CONFIG DATA TYPE, 32 bytes total. X: REAL Y: REAL Z: REAL W: REAL P: REAL R: REAL CONFIG_DATA: CONFIG
XYZWPREXT	An XYZWPREXT consists of three REAL components specifying a Cartesian location (x, y, z), three REAL components specifying an orientation (w, p, r), and a component specifying a configuration string. It also includes three extended axes, 44 bytes total. X: REAL Y: REAL Z: REAL W: REAL P: REAL R : REAL CONFIG_DATA: CONFIG EXT1: REAL EXT2: REAL EXT3: REAL
JOINTPOS	Consists of a REAL representation of the position of each axis of the group. Can not access directly to the fields. CNV_REL_JPOS and CNV_JPOS_REL built-ins can be used to access the real value.
CONFIG	Contains the following fields. CFG_TURN_NO1: INTEGER CFG_TURN_NO2: INTEGER CFG_TURN_NO3: INTEGER CFG_FILP: BOOLEAN CFG_LEFT: BOOLEAN CFG_UP: BOOLEAN CFG_FRONT: BOOLEAN
FILE	Save the information in the time of OPEN. You must use a FILE variable in OPEN FILE, READ, WRITE, CANCEL FILE and CLOSE FILE statement.

Data type is converted automatically in the case of assigning INTEGER variable to REAL one.
Data type is not converted automatically in the case of assigning REAL variable to INTEGER one. REAL value must be converted to INTEGER with TRUNC or ROUND built-in functions before assignment.

TRUNC(x) retunes the value of x after any fractional part has been removed. For example, if x = 2.5, the .5 is removed and a value of 2 is returned.

ROUND(X) returns the value of half adjust x. For example, if x = 2.5, a value of 3 is returned.

```
VAR
Int_val : INTEGER
Rel_val : REAL
```

```
BEGIN
   Rel_val = int_val
   Int_val = TRUNC( rel_val )
   Int_val = ROUND( rel_val )
END
```

In the case of adding 1mm of Z direction to POSITION variable, describe following.

```
VAR
p1 : POSITION
BEGIN
p1.location.z = p1.location.z + 1
END
```

In the case that assigning position variable, converting type is done automatically between different types. It allows converting from POSITION to XYZWPR and from JOINTPOS to POSITION.

If the program accesses to each field, describing as follows. This is the same case of structural variable and user type variables.

```
Variable = Variable_name.Field_name.Field_name…
Variable_name.Field_name.Field_name… = Variable.
VAR
old_config : CONFIG
tmp_turn      : INTEGER
BEGIN
tmp_turn = old_config.cfg_turn_no1    -- Assignment of structural field
old_config.cfg_turn_no1 = 5              -- Assigning the value to structural field
END AAA
```

3.2.8 User-Defined Data Types

KAREL allows defining structures as follows.

```
TYPE
person_t = STRUCTURE       -- Contain both INTEGER and BOOLEAN structure, person_t
      person_id    : INTEGER
      blood_type    : INTEGER
      man_flag      : BOOLEAN
ENDSTRUCTURE

ext_person_t = STRUCTURE
      person         : person_t -- Contain the user-defined structure, person_t
      phone_num    : INTEGER
ENDSTRUCTURE
```

It allows assigning structure variables. And structure allows using as an argument and a value of return the routine.

But varying length array and other indefinite size variables are not allowed to use as the field of the structure. Field of not itself structure allows containing.

3.2.9 Array

Up to 3 dimensions of array and varying length array are allowed defining as variables.

```
VAR
Array_name : ARRAY[length] OF Type_name
Array_name : ARRAY[length, length] OF Type_name
Array_name : ARRAY[length, length, length] OF Type_name
```

Following is the sample of declaration of array length is 10, INTEGER 1 dimension array, id10.

```
VAR
id10 : ARRAY[10] OF INTEGER
```

Declarations of item number 20 * 10, INTEGER 2 dimensions id20_10 is as follows.

```
VAR
id20_10: ARRAY[20,10] OF INTEGER
```

Accessing to each field is as follows.

```
id10[5] = 10
id20_10[18,7] = id10[3]
```

3.2.10 Constants

Constants are declared as follows.

```
CONST
AAA_C      = 111
CM2INCH_C = 2.5
PGNAME     = 'TEST'
```

These constants used in the program are changed to actual value in translating.

Hexadecimal numeral is not described directly in KAREL. If required, define like following constants easily viewable.

```
CONST
Ox000003FF =         1023
Ox0000FFFF =        65535
Ox00FF0000 =    16711680
Ox0FFFFFFF = 268435455
BEGIN
valuint = valuint AND Ox0FFFFFFF
```

WARNING
First character of a constant must be alphabet “O” not numeric zero,”0”.

3.2.11 Usable Operands

Following operators are allowed using operations of constants or variables or comparing.

Arithmetic operator : + - / * DIV MOD
Relational operator : < <= = <> >= >
Boolean operator : AND OR NOT

AND, OR, NOT of Boolean operators do logical operation (result is TRUE or FALSE) with BOOLEAN type variable and do bit operation (operating each bit, result is integer) with INTEGER type variable.

Other particular operator is as follows.

>=< : Relational operator of position data.
Determines if two POSITION operands are approximately equal and produces a BOOLEAN result. Acceptable value of judgment is allowed specifying in the program.
: : Operand of multiplication of position matrix. It allows that Matrix = Matrix : Matrix.
INV built-in function is allowed inverse matrix.
: Operand of scalar product. Scalar = Vector # Vector.
@ : Operand of vecrot product. Vector = Vector @ Vector.

3.2.12 Structure Statement

Following structure statement is allowed using in KAREL.

IF Statement

```
IF Boolean THEN
    Instruction -- execute while Boolean = TRUE
ENDIF

IF Boolean THEN
    Instruction -- execute while Boolean = TRUE
ELSE
    Instruction -- execute while Boolean = FALSE
ENDIF
```

SELECT Statement

```
SELECT variable OF
    CASE(value1):
    Instruction
    CASE（value2, value3, value4, …）:  -- Enable to specify multi number
    Instruction
    CASE（value5, value6）:
    Instruction
ELSE: -- None of above case. Alarm occurs if ELSE is not described.
    Instruction
ENDSELECT
```

FOR Statement

```
FOR variable = initial_value TO final_value DO   -- Even if initial_value = final_vale, loop is once executed.
    Instruction
ENDFOR

FOR variable = initial_value DOWNTO final_value DO -- DOWNTO is used if decrease the count
    Instruction
ENDFOR
```

WHILE Statement

```
WHILE Boolean DO -- Instruction is never executed if Boolean = FALSE at first.
```

```
    Instruction
ENDWHILE
```

REPEAT Statement

```
REPEAT
    Instruction
UNTIL Boolean -- Even if Boolean = TRUE, instruction is executed once.
```

GOTO Statement

```
GOTO lavel_name -- GOTO must be executed correct position.
lavel_name::
```

GOTO Statement is not allowed jumping into other loop (FOR, REPEAT, WHILE) and from other loop.

3.2.13 Routine Call

```
********************************************************************************
Here is a sample to define and call a routine that has no argument and return value. Routine name is
my_routine.
********************************************************************************
PROGRAM example

ROUTINE my_routine              -- Definition of my_routine. No argument and return value.
BEGIN
    WRITE( 'now in my_routine', CR ) -- Displaying characters on USER screen on TP with WRITE
    statement
END my_routine                  -- CR is line feed code
BEGIN
    WRITE( 'start example', CR )
    my_routine                  -- Calling my_routine here
END example
```

If it has no argument, only describing routine name at the calling.

```
********************************************************************************
Here is a sample to define and call a routine that has arguments and return value. Routine name is
square().
********************************************************************************
PROGRAM example
VAR
    tmp_int:INTEGER
ROUTINE square( param_int : INTEGER ) -- an argument, no return value
BEGIN
    param_int = param_int * param_int    -- Squares specified argument
END square
BEGIN
    tmp_int = 2
    square( tmp_int )                   -- tmp_int is squared by call-by-reference
  WRITE( 'tmp_int * tmp_int = ',tmp_int, CR )
END example
```

In this case tmp_int is both input and output for the routine.

```
********************************************************************************
```
Here is a sample to define and call a routine that has arguments and return value. Routine name is multi_x().
```
********************************************************************************
PROGRAM example
VAR
    result_val:INTEGER
-- A routine to return the value of multi_num times power of soc_val.
ROUTINE multi_x ( soc_val : INTEGER; multi_num : INTEGER ) : INTEGER
VAR
    work_val : INTEGER    -- Declaration of local variable
    i        : INTEGER    -- Declaration of local variable
BEGIN
    work_val = 1
    FOR i = 1 TO multi_num DO          -- Even if multi_num = 1, loop is executed once.
    work_val = work_val * soc_val
    ENDFOR
    RETURN ( work_val )                -- work_val is returned as return value
END multi_x
-- main program
BEGIN
    result_val = multi_x( 2, 3 )       -- returns third power-of-2
    WRITE( 'result = ', result_val, CR )
END example
```

Return value type is described after name and argument divided with ":" in routine declaration (there is no need if required no return value). And in the case that there are multi arguments of routine, they are divvied with ";" in declaration.

```
********************************************************************************
```
There are call-by-reference and pass-by-value ways to specify arguments.
```
********************************************************************************
PROGRAM test
VAR
    item_num : INTEGER
    item_val : REAL
ROUTINE test_1( item:INTEGER; val:REAL )
BEGIN
  :
END test_1
BEGIN
    item_num = 1
    item_val = 2.5
    Test_1( item_num, item_val )
```

Call-by-reference is the way to specify the argument directly like above. In the case of call-by-reference, the value of called program is changed when it is assigned to arguments in the routine. For above example, the variable, item_num, of program test is changed to 5, when doing assignment of item = 5 in the routine, test_1. Please notice and write program or unexpected result occurs.

Pass-by-value is the way to specify arguments in parentheses like following.

```
    Test_1( (item_num), (item_val) )
```

In the case of pass-by-value, the variable value of called program is not changed even though it does assignment in the routine.

In the case of calling KAREL program as a sub program from TP program with arguments, they are not specified from KAREL program side. KAREL program get the argument with GET_TPE_PRM built-in function. Following sample is to call KAREL main program as sub program with argument.

```
PROGRAM PNS0010
1: R[2] = 2.5
2: CALL cal_reg( 1, R[2] )
    :
[ END ]

PROGRAM cal_reg
%NOLOCKGROUP
CONST
      prm_int   = 1
      prm_real = 2
      prm_str   = 3
VAR
      item          : INTEGER
      val           : REAL
      data_type : INTEGER
      dmy_int     : INTEGER
      dmy_real    : REAL
      dmy_str     : STRING[1]
      stat          : INTEGER
BEGIN
      -- First argument value
      GET_TPE_PRM( 1, data_type, item, dmy_real, dmy_str, stat )
      IF stat = 0 THEN
-- Check argument type
            IF data_type <> prm_int THEN
                  WRITE TPERROR( 'Illegal type argument', 1, data_type, cr)
                  ABORT
            ENDIF
      ELSE
            WRITE TPERROR( 'Illegal argument ', 1, stat, cr)
            ABORT
      ENDIF

-- Second argument value
      GET_TPE_PRM( 2, data_type, dmy_int, item_val, dmy_str, stat )
      IF stat = 0 THEN
-- Check argument type
      IF (data_type <> prm_int) AND (data_type <> prm_real) THEN
            --Error of teach pendant
            WRITE TPERROR( 'Illegal type argument ', 2, data_type, cr)
            --Abort
            ABORT
      ENDIF
ELSE
      WRITE TPERROR( 'Illegal argument ', 2, stat, cr)
      ABORT
ENDIF
```

END cal_reg

See KAREL Reference Manual to refer built-in function of GET_TPE_PRM.

3.2.14 Global / Local variables

3.2.14.1 Global variables

Global declarations are recognized throughout a program.

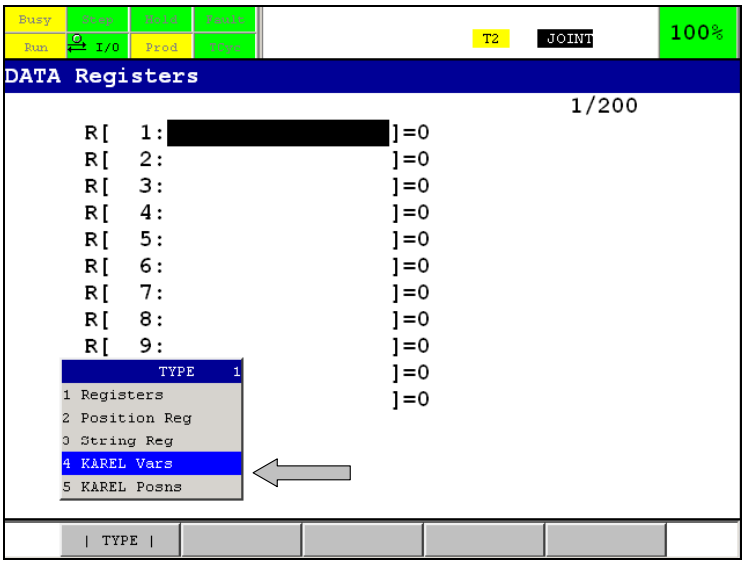
- Global declarations are referred to as static because they are given a memory location that does not change during program execution, even if the program is cleared or reloaded (unless the variables themselves are cleared.)
- Declarations made in the main program, as well as predefined identifiers, are global.
- The scope rules for predefined and user-defined routines, types, variables, constants, and labels are as follows:
- All predefined identifiers are recognized throughout the entire program.
- Routines, types, variables, and constants declared in the declaration section of a program are recognized throughout the entire program, including routines that are in the program.

The global variable defined in program test_1 and test_2 are allowed using in program test_3 to define like following.

```
PROGRAM test_1
%CMOSVARS
VAR
   Val_1 : INTEGER

PROGRAM test_2
VAR
   Val_2 : REAL

PROGRAM test_3
VAR
   Val_1 IN CMOS FROM test_1 : INTEGER
   Val_2 IN DRAM FROM test_2 : REAL
```

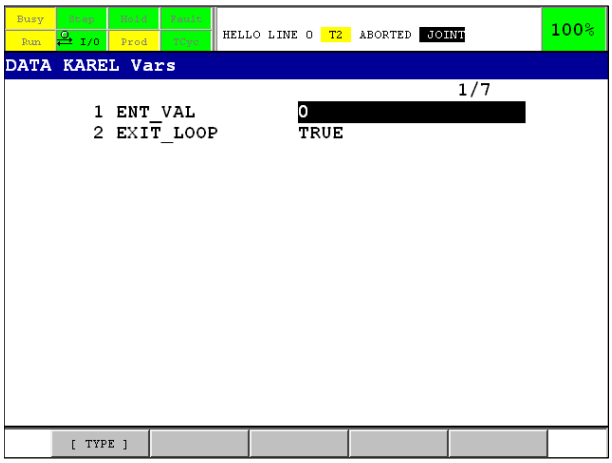
IN CMOS means that the specified variable is defined in CMOS. IN DRAM means that the variable is defined in DRAM. DRAM data is lost when cycle power.

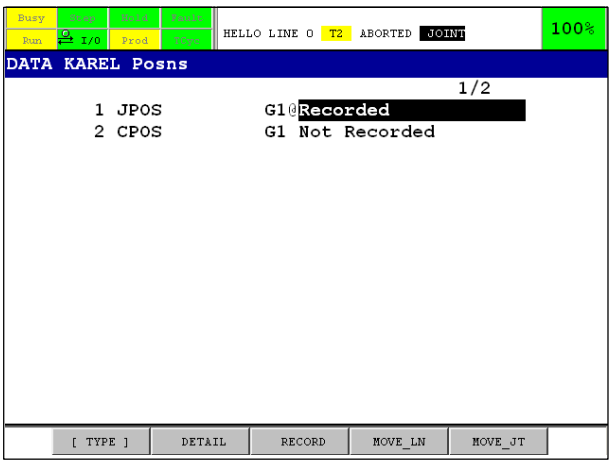
Global variable values are allowed to see from teach pendant. Select DATA screen and select KAREL Vars or KAREL Posns from F1 menu.



KAREL Vars

KAREL Posns





Using global variable between KAREL programs gets easier about giving and taking the data. However there is possibility to refer or set the data simultaneously from multiple locations in the KAREL program, then it causes some problems such as the KAREL program does not work well.
Please be careful very much about the date treatment when you use global variables.

3.2.14.2 Local Variables

The following rules apply to local declarations and definitions:

- Local declarations are recognized only within the routines where they are declared.
- Local data is created when a routine is invoked. Local data is destroyed when the routine finishes executing and returns.
- The scope rules for predefined and user-defined routines, variables, constants, and labels are as follows:
- Variables and constants, declared in the declaration section of a routine, and parameters, declared in the routine parameter list, are recognized only in that routine.
- Labels defined in a program (not in a routine of the program) are local to the body of the program and are not recognized within any routines of the program.
- Labels defined in a routine are local to the routine and are recognized only in that routine.
- Types cannot be declared in a routine, so are never local.

Local variables are limited to use only its routine. It is safer to use local variables if they are only use in its routine. For example in case of using a global variable as a loop counter for FOR statement, and calling other routine from its FOR statement, and its variable is used as a loop counter in the FOR statement in called routine, then FOR statement works well in last called routine, but FOR statement of first calling routine does not work normally.

```
PROGRAM bad_i
VAR
    i : INTEGER -- Loop counter

ROUTINE disp_num( cnt:INTEGER )
VAR
    wrk : INTEGER
BEGIN
    wrk = cnt
    FOR i = 1 TO 5 DO          -- Using global variable, i, as a loop counter
    WRITE( 'CNT', wrk, CR) -- In this case, calling this routine from main program FOR statement,
    wrk = wrk + 1              -- it works abnormal.
    ENDFOR
END disp_num

BEGIN
    FOR i = 1 TO 10 DO -- Using global variable, i, as a loop counter
    disp_num( i )
    ENDFOR
END bad_i
```

WARNING
Local variables are not referred from DATA screen because it is created when a routine is invoked

3.2.15 User and Tool Frame

In KAREL instruction associated with position data, the values of system variable $GROUP[n].$FRAME and $GROUP[n].$UTOOL (n: group number) are used as user frame and tool frame. These frame values are needed to set adequate value by KAREL program before executing instruction associated with position data.

Current user frame and tool frame are used in TP program motion. In case of using this user frame and tool frame in KAREL program, it is necessary to set the value of current user and tool frame to $GROUP[n].$FRAME and $GROUP[n].$UTOOL.

```
VAR
    uframe_num : INTEGER
    utool_num  : INTEGER
    pos1 : XYZWPR IN GROUP[1]

BEGIN
uframe_num = $MNUFRAMENUM[1]    -- Current user frame number
$GROUP[1].$FRAME = $MNUFRAME[1, uframe_num] -- Assign the value of current user frame to
                                                --user frame used in KAREL

utool_num = $MNUTOOLNUM[1]          -- Current tool frame number
```

$GROUP[1].$UTOOL = $MNUTOOL[1, utool_num] -- Assign the value of current tool frame to tool --frame used in KAREL

Taught user or tool frame number is not same of KAREL position data. It is different from TP program. Then KAREL position data is user and tool frame that are set in $GROUP[n].$FRAME and $GROUP[n].$UTOOL at the time of using.

Following is the way to set the value of world frame to user and tool frame of KAREL.

$GROUP[1].$FRAME = $MOR_GRP[1].$NILPOS
$GROUP[1].$UTOOL = $MOR_GRP[1].$NILPOS

$NILPOS is always set the value of world frame.

3.2.16 Input/Output Signals of KAREL

Following is comparing of the signals of using KAREL with TP program.

Signal Name in KAREL	Data Type	Signal Name in TP	Corresponding Signal
DIN	BOOLEAN	DI	Digital I/O (Input)
DOUT		DO	Digital I/O (Output)
RDI		RI	Robot I/O (Input)
RDO		RO	Robot I/O (Output)
GIN	INTEGER	GI	Group I/O (Input)
GOUT		GO	Group I/O (Output)
AIN		AI	Analogue I/O (Input)
AOUT		AO	Analogue I/O (Output)
OPIN	BOOLEAN	―	Panel I/O (Input)
OPOUT		―	Panel I/O (Output)
TPOUT		―	Teach Pendant LED Output

Operating each signal is as follows.

```
%INCLUDE kliotyps --I/O type definition

IF (DIN[1] = ON) AND (RDI[1]=OFF) THEN
   DOUT[10] = ON
   RDO[1]   = ON
ELSE
   DOUT[10] = OFF
   RDO[1]   = OFF
ENDIF

gin_val : INTEGER

gin_val = GIN[1] * 2
GOUT[1] = gin_val
```

How to pulse output is following.

```
PULSE DOUT[1] FOR 100
PULSE DOUT[2] FOR 100 NOWAIT
PULSE DOUT[3] FOR 100
```

If NOWAIT is specified in a PULSE statement, the next KAREL statement will be executed concurrently with the pulse.
If NOWAIT is not specified in a PULSE statement, the next KAREL statement will not be executed until the pulse is completed.

Relation of AIN/AOUT and actual analogue output (plus or minus 10V) is different from used I/O instrument. Contact FANUC for more information of corresponding value.

OPIN/OPOUT is allowed referring panel I/O (SI/SO) and peripheral I/O (UI/UO).

OPIN	SI	Comments (R-30*i*A)	OPOUT	SO	Comments (R-30*i*A)
0	0	－	0	0	REMOTE（*1）
1	1	FAULT RESET	1	1	BUSY（*1）
2	2	REMOTE	2	2	HELD（*1）
3	3	*HOLD	3	3	FAULT
4	4	USER1	4	4	BTAL（*1）
5	5	USER2	5	5	－
6	6	CYCLE START	6	6	－
7	7	－	7	7	TPENBL（*1）
8	8	CE/CR Select b0	8	8	－
9	9	CE/CR Select b1	9	9	－
10	10	－	10	10	－
11	11	－	11	11	－
12	12	－	12	12	－
13	13	－	13	13	－
14	14	－	14	14	－
15	15	－	15	15	－

（*1） These signal is not on the operating box.

OPIN	UI	Comments (R-30*i*A)	OPOUT	UO	Comment (R-30*i*A)
16	1	*IMSTP	16	1	CMDENBL
17	2	*HOLD	17	2	SYSRDY
18	3	*SFSPD	18	3	PROGRUN
19	4	CSTOPI	19	4	PAUSED
20	5	FAULT RESET	20	5	HELD
21	6	START	21	6	FAULT
22	7	－	22	7	ATPERCH
23	8	ENBL	23	8	TPENBL
24	9	RSR1/PNS1	24	9	BATALM
25	10	RSR2/PNS2	25	10	BUSY
26	11	RSR3/PNS3	26	11	ACK1/SNO1
27	12	RSR4/PNS4	27	12	ACK2/SNO2
28	13	RSR5/PNS5	28	13	ACK3/SNO3
29	14	RSR6/PNS6	29	14	ACK4/SNO4
30	15	RSR7/PNS7	30	15	ACK5/SNO5
31	16	RSR8/PNS8	31	16	ACK6/SNO6
32	17	PNSTROBE	32	17	ACK7/SNO7
33	18	PROD_START	33	18	ACK8/SNO8
34	19	－	34	19	SNACK
35	20	－	35	20	－

Referring peripheral I/O form OPIN/OPOUT, it is allowed referring by adding 15 to UI/UO number.

WARNING
Because system software output OPOUT value for necessity you must not change it from KAREL.

TPOUT allows you tot see teach pendant LED status.

TPOUT	LED Comment (R-30*i*A)
1	FAULT
2	HOLD
3	STEP
4	BUSY
5	RUNNING
6	(Depends on application)
7	(Depends on application)
8	(Depends on application)
9	JOINT
10	XYZ
11	TOOL

WARNING
Because system software also output TPOUT value for necessity you must not change it from the KAREL program.

3.2.17 Condition Handlers

The condition handler feature of the KAREL language allows a program to respond to external conditions more efficiently than conventional program control structures allow. It is like PLC function worked on system software. Condition handlers allows a KAREL program that monitors some status to be executed with TP program simultaneously, it is needed to realize like PLC function with KAREL. Following is the explanation of condition handlers usage.

3.2.17.1 Monitor Block

Monitor block defines a request that is “monitor DI[1], if it turns to ON, then turn on DO[12]” to condition handler with the format of “monitored thing + then what to do”.

Monitor block is always defined as “ if A, do B”. Hereinafter we call A is condition, and B is action. And doing action after A, it is called trigger.

Only creating monitor block in KAREL program, nothing happens. You must request to condition handler to scan monitored condition. If no request is received, scan is never done. In short, it is necessary to enable the monitor block.

It is called “ENABLE” to enable the monitor block. “DISABLE” is to disable the monitor block. After disabling the monitor block, the condition handler stops the scan.

“PURGE” is the operation of to delete the created monitor block.
KAREL programming level allows following monitor block operations. Then it is KAREL instruction.

- Create
- ENABLE
- DISABLE
- PURGE

To use the usage of “if turned ENABLE, do scanning until trigger” is called global condition handler. It is automatically turned DISABLE once triggered. Scanning is not done until monitor turns to ENABLE in the program. It is not PURGE operation, then the monitor block can be turned to ENABLE.

Following is the sample of global condition handler that DO[10] is turned to OFF if DO[1] is turned to ON.

```
CONDITION[1]:
  WHEN DIN[1] = ON DO
  DOUT[10] = OFF
ENDCONDITON
```

3.2.17.2 Global Monitor

Following is the sample to create global monitor.

```
-- Creatation
CONDITION[i]:                -- i is the value of identifier of the monitor
  WHEN condition1 DO         -- condition1 is the condition such as DIN[1] = ON
    action1
    action2
  WHEN condition2 DO         -- action1 is the action such as DOUT[1] = ON
    action3
    ENABLE CONDITION[1]       -- Enable CONDITION[i] again
ENDCONDTION[i]               -- Complete

-- ENABLE
ENABLE CONDITION[i]          --Start scanning

-- DISABLE
DISABLE CONDITION[i]         -- Never to scan

-- PURGE
PURGE CONDITION[i]           -- Deleting CONDITON[i]
```

Once triggered, global condition handler is automatically turned to DISABLE. Like above sample of condition2, enabling the condition, once enable, it is enabled at all times.

And creating global monitor must be given own number. But it is OK to not use own number if no possibility is assumable that two conditions are not overlapped while program running in same time.

Therefore it can be executed to repeat create and PURGE operation of CONDITION[1]. Alarm is occurred if new CONDITION[1] is about to create before PURGE it.

No alarm or no event is occurred if ENABLE or DISABLE or PURGE operation is executed against to nonsexist global condition. No alarm is occurred when the operation to already ENABLE or DISABLE is executed.

One monitor block can have multi conditions in the time of monitor creation. And each condition can have multi action. In this case all conditions are DISABLE after one of them is triggered.

3.2.17.3 Global Condition

Following is the sample allowed to be specified condition as global monitor.

```
WHEN DOUT[1]=ON DO    ...      -- Trigger if DOUT[1] is ON
                                -- DOUT can change DIN etc.
WHEN DOUT[1]=OFF DO ...        -- Trigger if DOUT[1] is OFF.
WHEN DOUT[1]+ DO ...           -- Trigger if DOUT[1] is turned to ON from OFF.
WHEN DOUT[1]- DO ...           -- Trigger if DOUT[1] is turned to OFF from ON.
WHEN var1 = var2 DO ...        -- Trigger if var1 equals to var2
                               -- "=" is OK if {<>,<,<=,>,>=}
                               -- It is OK if variable or constant is GIN[i]
WHEN ERROR[n] DO ...           -- Trigger if specified error occurred
                              -- ERROR[*] is OK
WHEN ABORT DO ...              -- Trigger if abort
WHEN PAUSE DO ...              -- Trigger if pause
WHEN CONTINUE DO ...           -- Trigger if continue
```

There are other conditions such as events. Only INTEGER can be used. BYTE and SHORT are never used for variables.

It allows one condition to connect each condition with AND/OR. However, AND and OR cannot be existed in the one condition.

Following is the sample of trigger if all of DIN[1-3] is ON or one of DIN[4-6] is ON.

```
CONDITION[1]:
   WHEN DIN[1] AND DIN[2] AND DIN[3] DO
      DOUT[1] = ON
   WHEN DIN[4] OR DIN[5] OR DIN[6] DO
      DOUT[2] = ON
ENDCONDITION
```

3.2.17.4 Actions

Following are the samples that are allowed to be specified actions.

```
variable = (variables, constants, DIO)    -- Assignment to variable
DOUT = (variables, constants)             -- I/O output
CANCEL                                    -- Cancel of motion
Routine Call                              -- Routine call (ISR: describe after)
                                          -- No arguments
ENABLE CONDTION[index]                    -- ENABLE monitor that includes itself
                                          -- It makes itself keep ENABLE.
DISABLE CONDITION[index]                  -- DISABLE monitor
PULSE DOUT[n] FOR time                    -- Pulse output
ABORT                                     -- Abort the program
PAUSE                                     -- Pause the program
```

Others are CONTINUE action, signal event etc.
Only INTEGER can be used. BYTE and SHORT are never used for variables.

3.2.17.5 Routine Call (ISR)

Specifying interrupt routine (Interrupt Service Routine) to action of condition handler, specified routine are allowed executing when condition is triggered and interrupt the running KAREL program.

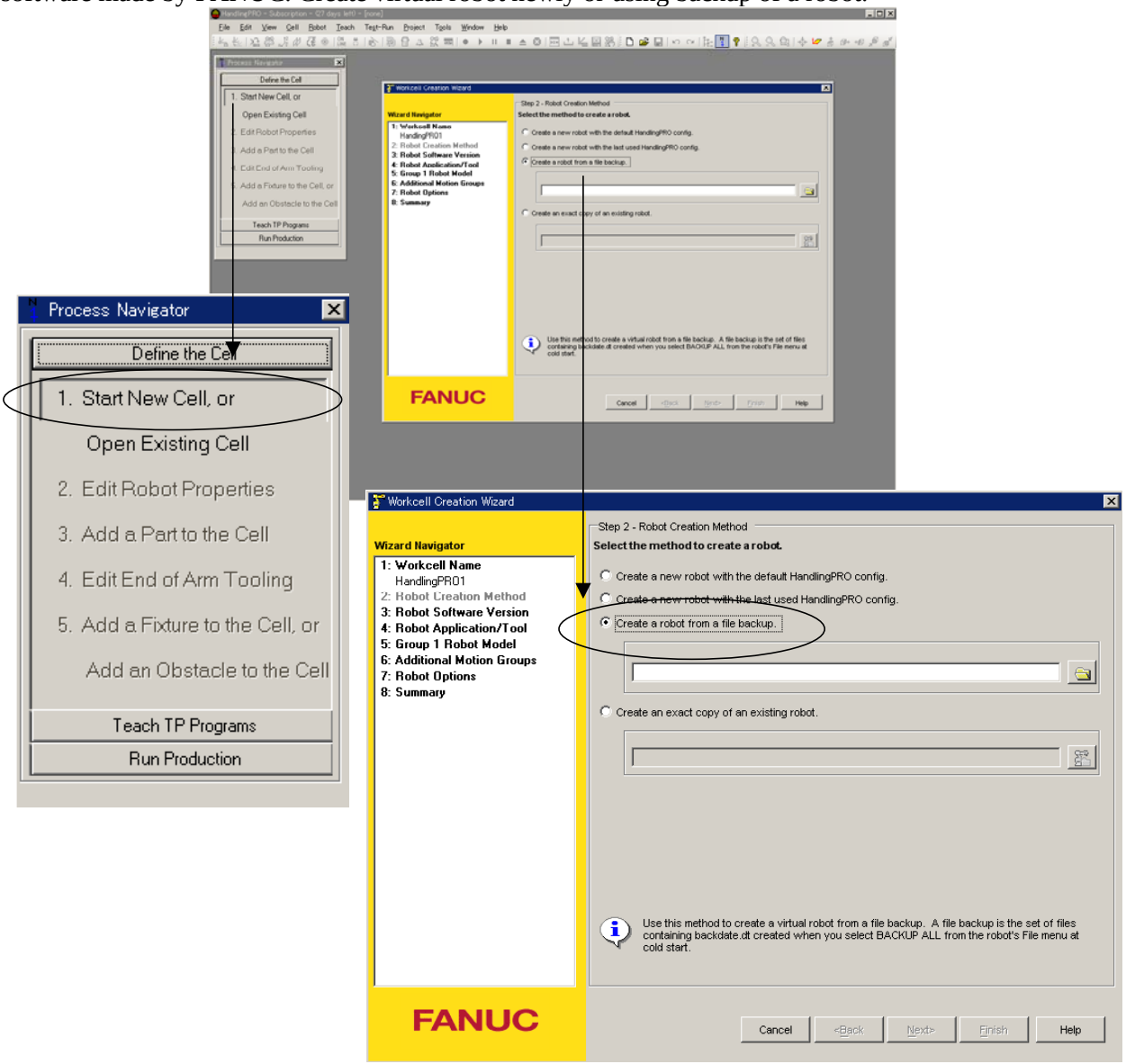
```
ROUTINE my_routine
BEGIN
   trig_cnt = trig_cnt + 1
   IF trig_cnt = 10 THEN
   DOUT[1] = ON
ELSE
   ENABLE CONDITION[1]
ENDIF
END my_routine
 :
 :
CONDITION[1]:
   WHEN DIN[1] DO
      my_routine
ENDCONDITION
ENABLE CONDITION[1]
```

Executing above, local routine of my_routine is called when DIN[1] is ON, and counting the trigger, DOUT[1] is output if it triggers 10 times. Action of condition handler can be restrictive treatment, but specifying interrupt routine gets you use all KAREL functions. However, when an interrupt routine is started, the interrupted KAREL program is suspended until the routine returns, then the KAREL program gets insecurity if interrupt routine that does a lot of treatment creates.

Interrupt routine is very useful, but you must pay attention about this point.

4 CREATE KAREL PROGRAMS

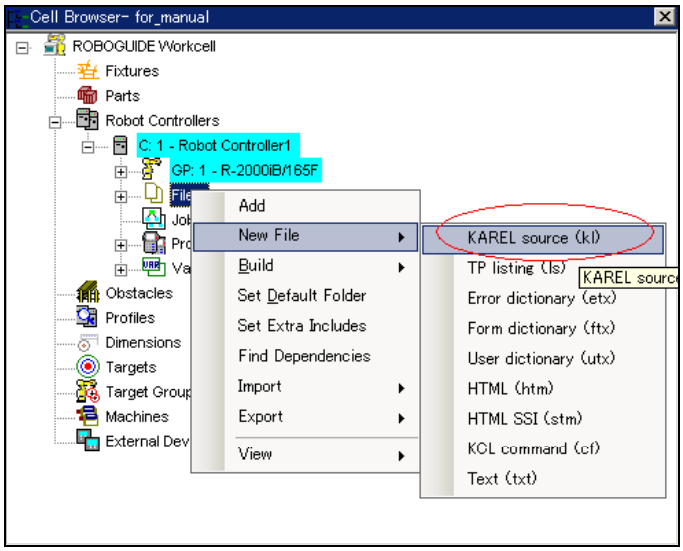
Following is an explanation of creating KAREL programs. It is necessary to use ROBOGUIDE that is PC software made by FANUC. Create virtual robot newly or using backup of a robot.



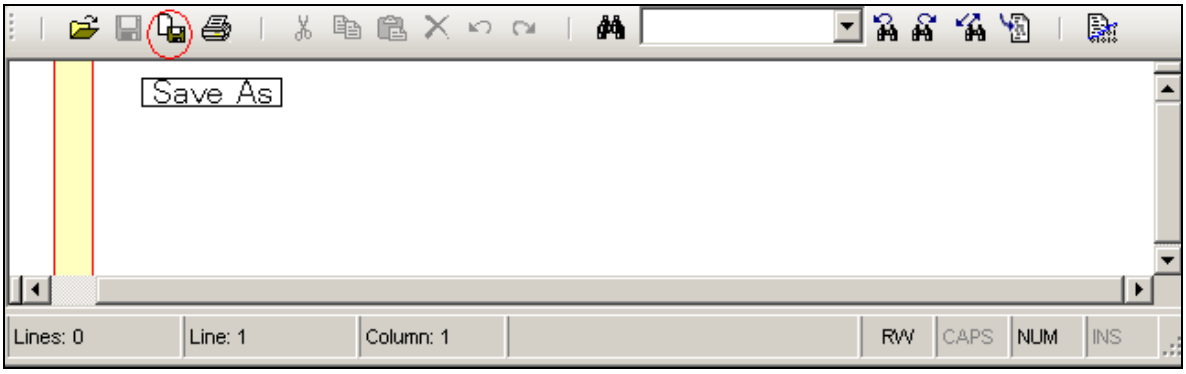
4.1 TRANSLATING OR ADDING KAREL FILES FOR ROBOGUIDE

4.1.1 The Sample of Newly Adding a KAREL Program

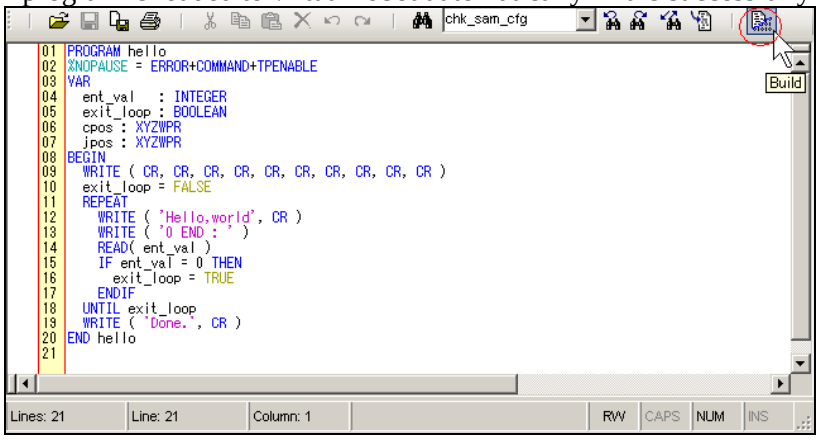
Do right click on cell browser, select new file and “KAREL source”. Select “Add” in case of adding created KAREL file. Following is a sample. (If cell browser is not displayed, select from menu.)



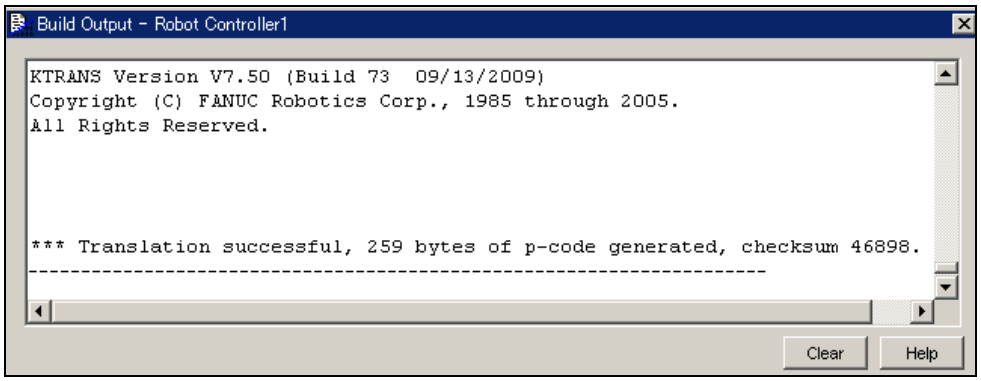
Adding newly file is empty. It is named like “Untitled1.kl”. Click “Save As” button and input KAREL file name within 8 characters such as formtest.kl.



After inputting the KAREL program, click upper right build button, then translating the KAREL program. Translated KAREL program is loaded to virtual robot automatically if it is successfully built.

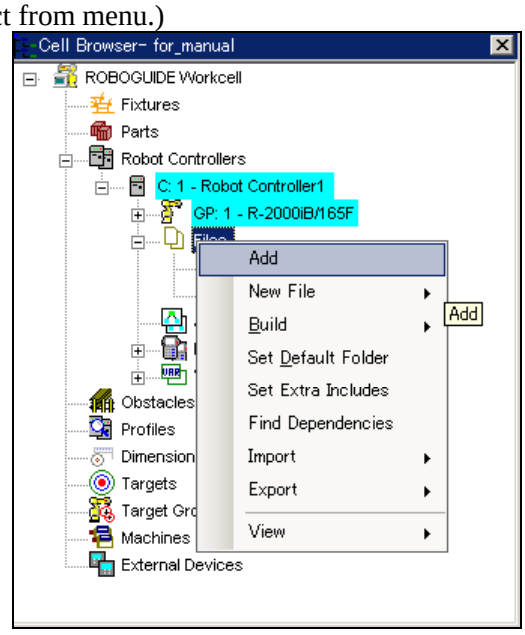


Result of build is displayed on other window. Simple explanation is described if some errors are occurred.

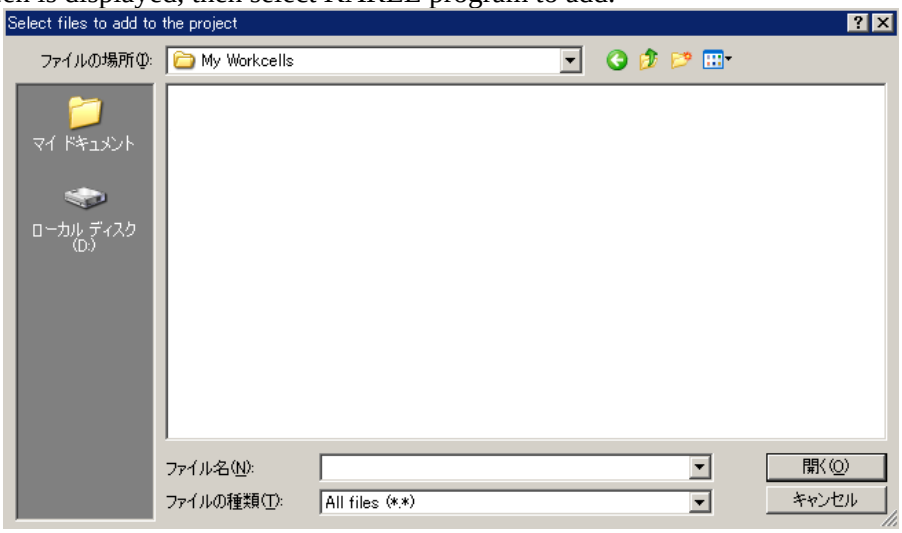


4.1.2 The Sample of Adding Existed KAREL Program

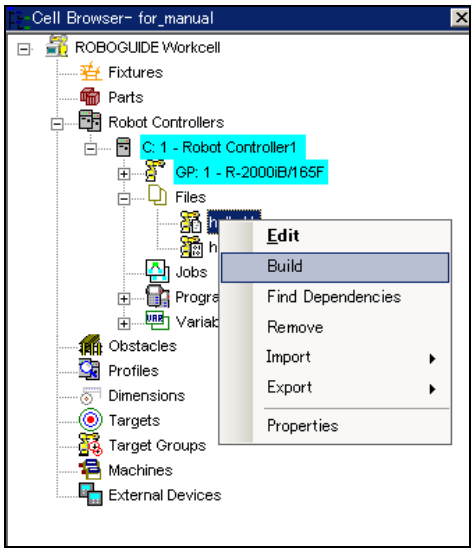
Do right click on file of cell browser and select “Add”. Following is the displaying sample. (If cell browser is not displayed, select from menu.)



Following screen is displayed, then select KAREL program to add.



Do right click on added file and select build. Then KAREL program is translated. In following case, getattr.pc is created after build. Translated KAREL program is loaded to virtual robot automatically if it is successfully built.



Like “The Sample of newly adding a KAREL program”, KAREL program can be built

4.2 SYNTAX OF KAREL PROGRAMS

4.2.1 Basic Syntax

```
-----------------------------------------------------------------------------
PROGRAM sample    --Program Name
-----------------------------------------------------------------------------
%NOBUSYLAMP
%NOLOCKGROUP
%NOPAUSE = ERROR + COMMAND + TPENABLE
%NOABORT = ERROR + COMMAND
VAR -- Variable definition
   status           : INTEGER    -- Definition of status as INTEGER type
-----------------------------------------------------------------------------
BEGIN -- A part of program execution
-----------------------------------------------------------------------------

SET_INT_REG(1, 100, STATUS)

END sample -- Program Name
```

Add above sample.kl on ROBOGUIDE. Describe its program name first and end of the KAREL program.and attribute, environment variable. (They are defined as %**). VAR is the definition of variables. Between BEGIN and END is the execution part. From double “--“ to end of line is comments. This part is ignored when build it.

In above sample register 1 is set by SET_INT_REG built-in function. See KAREL reference manual, Appendix A to know detail.

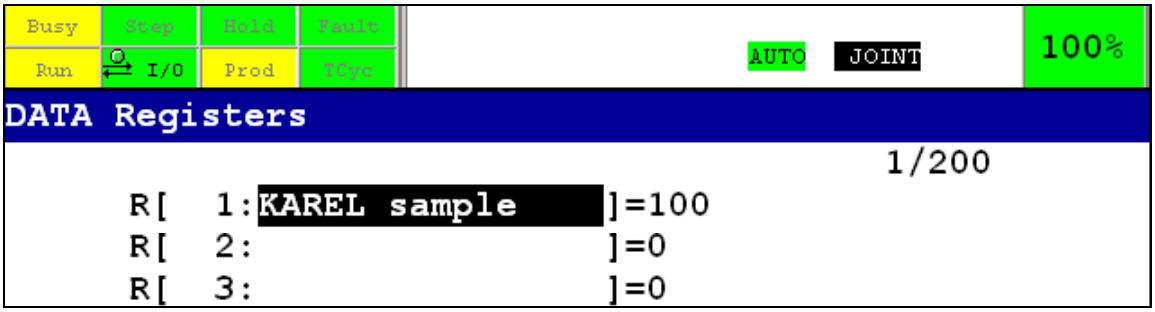
Changing above sample to following R[1] is turned to 100.

```
--------------------------------------------------------------------------------
PROGRAM sample
--------------------------------------------------------------------------------
%NOBUSYLAMP
%NOLOCKGROUP
%NOPAUSE = ERROR + COMMAND + TPENABLE
%NOABORT = ERROR + COMMAND

CONST
  REG_NUM_C = 1
  REG_VAL_C   = 100
VAR
  status          : INTEGER
--------------------------------------------------------------------------------
BEGIN
--------------------------------------------------------------------------------

SET_INT_REG(REG_NUM_C, REG_VAL_C, STATUS)

END sample
```

CONST means the definition of constants. Changing above sample.kl to following and build and execute sample.pc, R1 is changed to R1[KAREL sample] = 100.

```
--------------------------------------------------------------------------------
PROGRAM sample
--------------------------------------------------------------------------------
%NOBUSYLAMP
%NOLOCKGROUP
%NOPAUSE = ERROR + COMMAND + TPENABLE
%NOABORT = ERROR + COMMAND

CONST
  REG_NUM_C = 1
  REG_VAL_C   = 100
  K_SMPL        = 'KAREL sample'
VAR
  status          : INTEGER
--------------------------------------------------------------------------------
BEGIN
--------------------------------------------------------------------------------

SET_INT_REG(REG_NUM_C, REG_VAL_C, STATUS)
SET_REG_CMT(REG_NUM_C,K_SMPL,STATUS)

END sample
```

Check register on DATA screen after run the program, you can see the value is set.



SET_REG_CMT is built-in function to set comments to specified register. The same result can be got to set it like SET_REG_CMT(REG_NUM_C,'KAREL sample',STATUS)
To set strings in the KAREL program, you must enclose them with ‘’, single quotes.

CONST is the definition of constants. If in case of many defined constants are existed, then creates another KAREL file (sample2.kl), defines such as %include sample2 in the program, you can use constants described in sample2.kl

4.2.2 Program with Routine

It allows above program to describe as follows.

```
--------------------------------------------------------------------------------
PROGRAM sample
--------------------------------------------------------------------------------
%NOBUSYLAMP
%NOLOCKGROUP
%NOPAUSE = ERROR + COMMAND + TPENABLE
%NOABORT = ERROR + COMMAND
CONST
   REG_NUM_C = 1
   REG_VAL_C   = 100
   K_SMPL         = 'KAREL sample'
VAR
   STATUS          : INTEGER

--------------------------------------------------------------------------------
ROUTINE setreg
--------------------------------------------------------------------------------
BEGIN

SET_INT_REG(REG_NUM_C, REG_VAL_C, STATUS)
SET_REG_CMT(REG_NUM_C,K_SMPL,STATUS)

END setreg
--------------------------------------------------------------------------------
BEGIN -- A part of program execution
--------------------------------------------------------------------------------

setreg -- Call setreg routine

END sample
```

The routine setreg is called. The comment and the value are set to R[1]. Setreg does not have argument and return value.

For example, a program, adding in sequence from 1 to 10 and calculating result is output to R[2], is described as follows with routine.

```
--------------------------------------------------------------------------------
PROGRAM sample --Assign the sum from 1 to 10 to R[2]
--------------------------------------------------------------------------------
%NOBUSYLAMP
%NOLOCKGROUP
%NOPAUSE = ERROR + COMMAND + TPENABLE
%NOABORT = ERROR + COMMAND
CONST --Constant Definition
   START_NUM = 1
   FINISH_NUM    = 10
VAR --Variable Definition
   STATUS : INTEGER
   result : INTEGER
--------------------------------------------------------------------------------
ROUTINE sum( from_num: INTEGER; to_num:INTEGER) : INTEGER
--------------------------------------------------------------------------------
VAR
   idx : INTEGER
   add : INTEGER
BEGIN
     add = 0
     idx = 0
     FOR idx = from_num TO to_num DO
          add = add + idx --Assign argument to add
     ENDFOR
   RETURN(add)
END sum
--------------------------------------------------------------------------------
BEGIN -- Execute this program from here
--------------------------------------------------------------------------------
result = sum( START_NUM, FINISH_NUM) --Call routine and assign return value
                                                --to result
SET_INT_REG(2,result,STATUS)

END sample
```

Sum routine has two INTEGER arguments (from_num and to_num) and return INTEGER value (add).

To be executed this program sum is called and return value is assigned to “result”. It is output to R[2] with SET_INT_REG built-in function. “idx” and “add” defined in sum routine is enabled only in sum routine. If the same name variable is defined in other routine, there is no affection to sum routine. However, you must not define the same variable name in the routine defined as program variable In this case you must not define “STATUS” and “result” as routine variable.

4.2.3 The Sample of Using Condition Handler

Following is the sample to monitor the status of single step every 0.5 second and DO[1] is ON during single step. $SSR.$SINGLESTEP (BYTE type) value is got by built-in function is this sample.

NOTE
You must not change this system variable directly.

BYTE type variable are not allowed to specify in a condition, then it is monitored as follows. If timer is not used, the condition is always triggered, then some treatment might get damage. Therefore timer is used. The program is aborted when DO[10] is risen on (OFF -> ON).

```
---------------------------------------------------------------------------
PROGRAM chk_step
---------------------------------------------------------------------------
%SYSTEM
%NOBUSYLAMP
%NOLOCKGROUP
%NOPAUSE = ERROR + COMMAND + TPENABLE
%NOABORT = ERROR + COMMAND
%ENVIRONMENT iosetup
%INCLUDE kliotyps
CONST
   DO_SINGL = 1 --DO output number
VAR
   STATUS : INTEGER
   cond_id     : INTEGER
   exit        : BOOLEAN
   step_val : INTEGER
   on_off_time: INTEGER
   beat_timer:INTEGER
   entry :INTEGER
---------------------------------------------------------------------------
ROUTINE check_step
---------------------------------------------------------------------------
BEGIN
    --Output the value of $ssr.$singlestep to step_val
    GET_VAR(entry,'*system*','$ssr.$singlestep',step_val,STATUS)
    IF STATUS <> 0 THEN --GET_VAR returns not 0 to status if failed get the value.
        POST_ERR(STATUS,'',0,0)
    ELSE
        SET_PORT_VAL(io_dout,DO_SINGL,step_val,STATUS) --Output step_val
    value to DO[1].
    ENDIF
END check_step
---------------------------------------------------------------------------
BEGIN
---------------------------------------------------------------------------
   cond_id = 1
   on_off_time = 500
```

```
    step_val = 0

    CONDITION[cond_id]:
      WHEN on_off_time < beat_timer DO
        check_step
        beat_timer = 0
        ENABLE CONDITION[cond_id]
    ENDCONDITION
    ENABLE CONDITION[cond_id]
     cond_id = cond_id + 1

     CONDITION[cond_id]:
        WHEN DOUT[10]+ DO --Detect DO[10] rising edge
        exit = TRUE
        ENABLE CONDITION[cond_id]
     ENDCONDITION
     ENABLE CONDITION[cond_id]

    beat_timer = 0
    CONNECT TIMER TO beat_timer    -- Start timer.

    -- Wait forever
    exit = FALSE
    WAIT FOR exit = TRUE
    DISCONNECT TIMER beat_timer    -- Stop timer.

  END chk_step
```

GET_VAR built-in function is to get specified variable value.

Syntax : GET_VAR(entry, prog_name, var_name, value, status)

Input/Output Parameters:
[in,out] entry :INTEGER
[in] prog_name :STRING
[in] var_name :STRING
[out] value :Any valid KAREL data type
[out] status :INTEGER

%ENVIRONMENT Group :SYSTEM

Details:

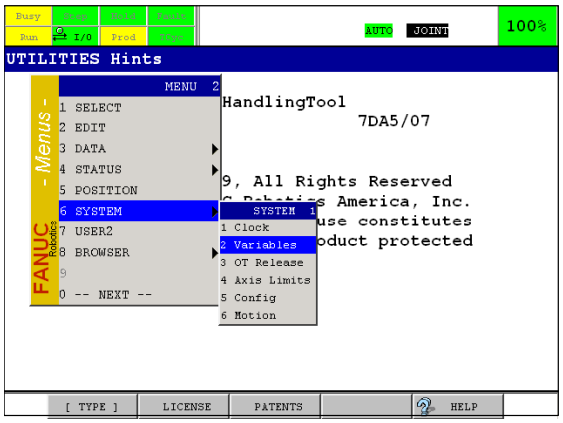
- entry returns the entry number in the variable data table of var_name in the device directory where var_name is located. This variable should not be modified.
- prog_name specifies the name of the program that contains the specified variable. If prog_name is blank, it will default to the current task name being executed. Set the prog_name to `*SYSTEM*' to get a system variable. prog_name can also access a system variable on a robot in a ring.
- var_name must refer to a static, program variable.
- var_name can contain field names, and/or subscripts.
- If both var_name and value are ARRAYs, the number of elements copied will equal the size of the smaller of the two arrays.

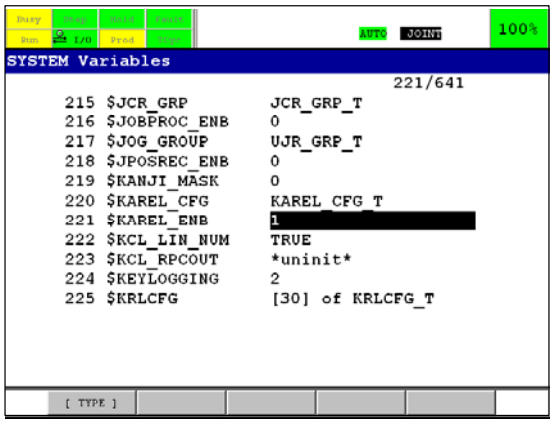
- If both var_name and value are STRINGs, the number of characters copied will equal the size of the smaller of the two strings.
- If both var_name and value are STRUCTUREs of the same type, value will be an exact copy of var_name .
- value is the value of var_name .
- status explains the status of the attempted operation. If not equal to 0, then an error occurred.
- If the value of var_name is uninitialized, then value will be set to uninitialized and status will be set to 12311.
- The designated names of all the robots can be found in the system variable $PH_MEMBERS[]. This also includes information about the state of the robot. The ring index is the array index for this system variable. KAREL users can write general purpose programs by referring to the names and other information in this system variable rather than explicit names.

4.3 KAEL EXECUTION

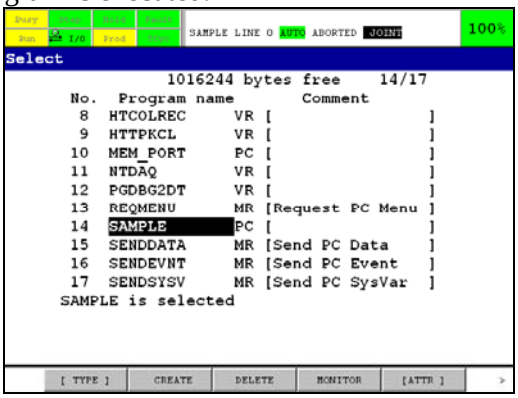
4.3.1 Execute from SELECT Screen

Set the system variable $KAREL_ENB to 1 from system variable screen (Menu -> NEXT -> SYSTEM -> Variables). Then KAREL programs are seen on SELECT screen.



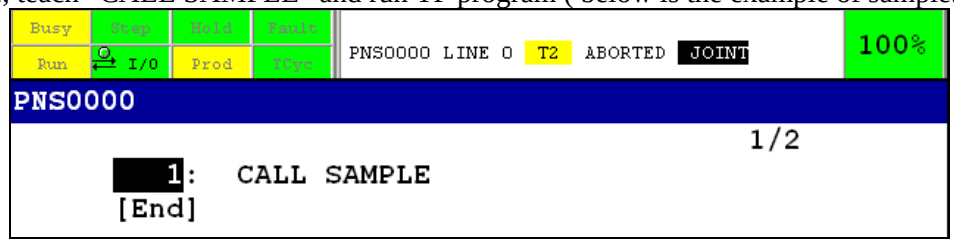


Display SELECT (Menu -> NEXT -> SELECT), F1[TYPE], select “ALL” or “KAREL Progs”, then KAREL programs are displayed. Set the cursor onto translated KAREL and push ENTER key, then KAREL program is selected (below figure is the sample when sample.pc is selected). Push “SHIFT” + “FWD” keys, then KAREL program is executed.



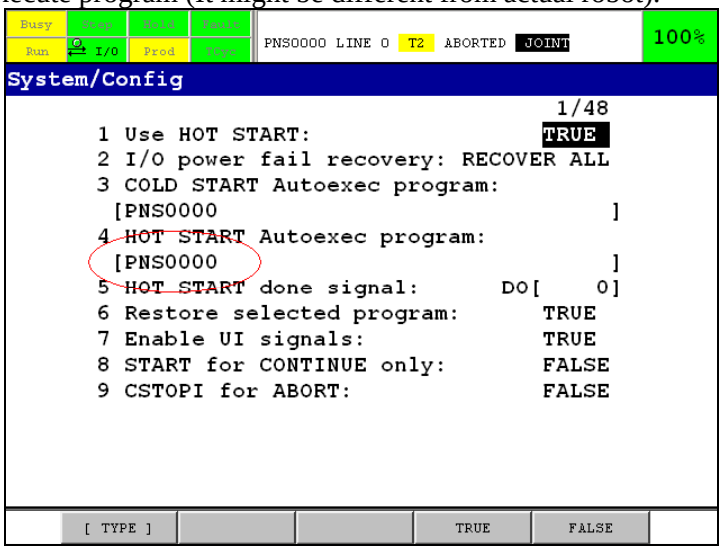
4.3.2 Execute from TP Program

For example, teach “CALL SAMPLE” and run TP program (below is the example of sample.pc)

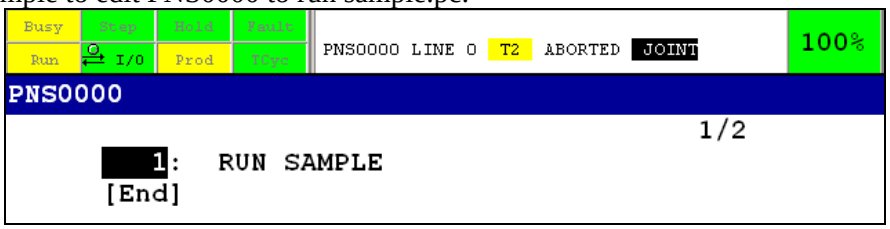


4.3.3 Execute by Auto Execute Program

Set “HOT START Autoexec program:” in system config screen to the program. Below is the sample that PNS0000 is set auto execute program (It might be different from actual robot).



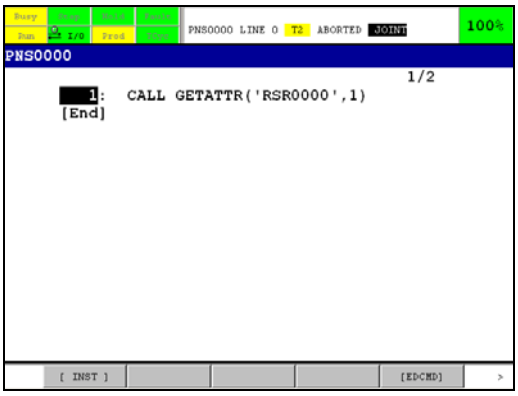
Below is the sample to edit PNS0000 to run sample.pc.



4.4 KAREL APPLICATION

4.4.1 The KAREL Sample to Monitor and Output the Program Protection

Teach TP program as follows. In below, RSR0000 is the name of TP program. Right side of RSR0000 is the register number.



Executing this program, R[1] is set to 100 if specified program (ex. RSR000) is not protected, or set to R[1] to 0 if it is protected.

Descript like follows:
GETATTR(‘PROGRAM NAME’, REGISTER NUMBER)

GETATTR(‘PROGRAM NAME’, 2) is to output to R[2] if whether it is protected or not.
GETATTR(‘PROGRAM NAME’, R[3]) is to output R[R3]]. Then if R[3] = 5, R[5] is the result of if protected or not.
GETATTR(‘PROGRAM NAME’, R[R[3]]) is to output R[10] when R[3] = 5 and R[5] = 10.

Warning is displayed if abnormal program or register number is specified.

WARNING
- The KAREL program is paused when TP program is paused such as releasing SHIFT key.
- Backup the memory card if it was deleted by miss operation and so on.
- KAREL program is not do power failure recovery. It always does from top of the program.

```
PROGRAM getattr
-------------------------------------------------------------------------
-- This program getarrr is get the attribute data from the specified
-- TP program and output the status to specified register
-------------------------------------------------------------------------

%NOLOCKGROUP
%NOPAUSE   = ERROR + COMMAND + TPENABLE
%NOABORT = ERROR + COMMAND + TPENABLE
%NOBUSYLAMP
%INCLUDE klevccdf
CONST
  TYPE_STRING = 3
  TYPE_INT      = 1
  PROTECT_OFF = 1
  PROTECT_ON   = 2

  NOT_PROTECT = 100
  PROTECTED    = 0
```

```
-- NOT_PROTECT is the value if not write protected. It is 100 here.
-- PROTECTED is the value if write protected.
-- REG_NUMBER is to output the status whether write protected or not

VAR
   param_no     : INTEGER
   data_type    : INTEGER
   int_val      : INTEGER
   real_val     : REAL
   str_val      : STRING[50]
   STATUS       : INTEGER
   prog_name    : STRING[50]
   reg_num      : INTEGER    --register number to be set

--------------------------------------------------------------------------
ROUTINE get_regnum : INTEGER
--------------------------------------------------------------------------
-- This routine get the second argument, reg_number, of
-- GETATTR('prog_name',reg_number) which is called from TP program.

VAR
   stat : INTEGER

BEGIN

   stat = 0
   -- Read second argument of getattr(File name, register number)
   param_no = 2
   GET_TPE_PRM(param_no, data_type, int_val, real_val, str_val, stat)
   --GET_TPE_PRM is to read arguments. In here it reads second one.
   --GET_TPE_PRM(param_no, data_type, int_value, real_value, str_value, status)
   --The value corresponding to data type is returned. If data type is INTEGER,
   --value is returned to int_val.
   IF (stat <> 0) THEN
      --Warning is posted if failed.
      POST_ERR(stat, '',0,0)
   ELSE
      IF ( data_type = TYPE_INT) THEN
        --Read register value is output to int_val. It is assigned to reg_num.
          reg_num = int_val    --Set the register number
      ENDIF
   ENDIF

   RETURN (stat)
END get_regnum
--------------------------------------------------------------------------
BEGIN -- Main program getattr
--------------------------------------------------------------------------
   -- Initialize variables
   param_no = 0
```

```
str_val = ''
prog_name = ''
reg_num = 0

-- Read first argument of getattr(File name, register number)
param_no = 1
GET_TPE_PRM(param_no, data_type, int_val, real_val, str_val, STATUS)
IF (STATUS <> 0) THEN
   --Warning is posted if failed.
   POST_ERR(STATUS, '',0,0)
ELSE
   IF ( data_type = TYPE_STRING ) THEN
      prog_name = str_val
      -- Read the attribute of TP program.
      GET_ATTR_PRG(prog_name, AT_PROTECT, int_val, str_val, STATUS)
      IF ( STATUS <> 0) THEN
         POST_ERR(STATUS, '',0,0)
      ELSE
         IF ( int_val = PROTECT_OFF ) THEN    -- Protect is off
            STATUS = get_regnum
            IF ( STATUS = 0 ) THEN
               -- Assigns to the register. The value (NOT_PROTECT) is assigned
               -- to specified register (reg_num). NOT_PROTECT value is
               -- defined first.
               SET_INT_REG(reg_num, NOT_PROTECT,STATUS)
               IF (STATUS <> 0) THEN
                  --Warning is posted if failed.
                  POST_ERR(STATUS,'',0,0)
               ENDIF
            ENDIF
         ELSE
            IF (int_val = PROTECT_ON) THEN      -- Protect is on
               STATUS = get_regnum
               IF ( STATUS = 0 ) THEN
                  -- Set the value to the specified register
                  SET_INT_REG(reg_num, PROTECTED,STATUS)
                  IF (STATUS <> 0) THEN
                     POST_ERR(STATUS,'',0,0)
                  ENDIF
               ENDIF
            ENDIF
         ENDIF
      ENDIF
   ELSE
    -- Wrong argument is specified.
    -- Switch USER screen and display the message.
    -- Force to switch USER screen.
      FORCE_SPMENU(TP_PANEL,SPI_TPUSER,1) -- Force the USER menu
 screen
      WRITE TPDISPLAY (CHR(cc_clear_win), CHR(cc_home))
```

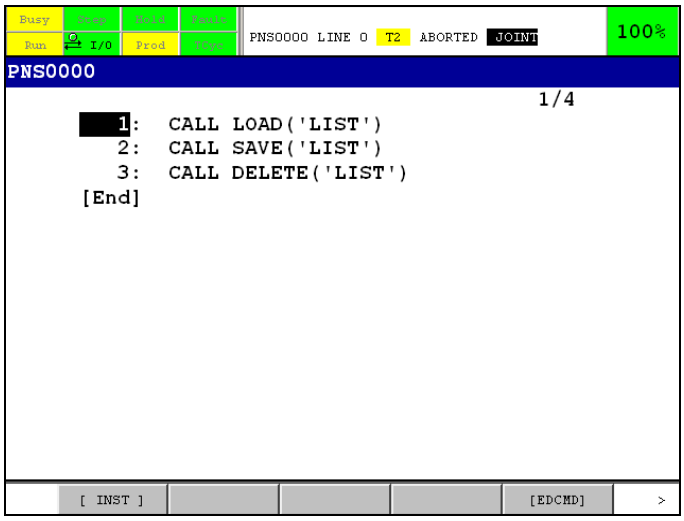
```
        WRITE TPDISPLAY ('Wrong program name was input.',CR)
        WRITE TPDISPLAY ('Please input correct program name.', CR,CR)
        WRITE TPDISPLAY ('GETATTR usage:',CR)
        WRITE TPDISPLAY ('GETATTR(program name, number)',CR)
      ENDIF
    ENDIF

  END getattr
```

4.4.2 Save, Delete, Load the Program Based on the List

Usage:
Load load.pc, save.pc and delete.pc to the robot controller.
Select MC: from file screen (Menu -> FILE) and switch the device (F5[UTIL] – Set Device). Below is the explanation in case of selected devices is MC:.(KAREL is allowed to execute from UD1:).

Put the program list (ex. LIST.DT) and TP programs in root of memory card. The extension of the list must be .dt.

Teach as follows. Specify argument (the name of list) to each KAREL program.



Below is the sample of list. Describe a TP program in every lines.

```
<MC:LIST.DT sample>
PNS0000.TP
PNS0001.TP
SAMPLE.TP
TEST.TP
```

WARNING
Last program of its list is not treated if no line feed is input to end of the list.

LOAD.PC
Loading programs according as the list specified for argument.

SAVE.PC
Saving programs to memory card according as the list specified for argument.

DELETE.PC
Deleting programs from the robot controller according as the list specified for argument.

Warning is posted when the listed program is not existed, or abnormal argument is specified .

＜load.kl＞

```
PROGRAM load

-------------------------------------------------------------------------
-- This program load is to load the TP program from specified
-- file list.
-- Usage:
-- Call load(filelist)
-------------------------------------------------------------------------

%NOLOCKGROUP
%NOPAUSE   = ERROR + COMMAND + TPENABLE
%NOABORT = ERROR + COMMAND + TPENABLE
%NOBUSYLAMP

CONST
   TYPE_STRING = 3

VAR
   status        : INTEGER
   param_no     : INTEGER
   data_type    : INTEGER
   int_val       : INTEGER
   real_val     : REAL
   str_val       : STRING[50]
   filelist     : FILE
   entry         : INTEGER
   dev_name     : STRING[10]
   list_name    : STRING[60]
   oneprog      : STRING[40]

-------------------------------------------------------------------------
BEGIN
-------------------------------------------------------------------------

   status = 0
   str_val = ''
   dev_name = ''
   list_name = ''
   oneprog = ''
   param_no = 1

   -- get parameter 1 that is the list of file to be loaded.
   GET_TPE_PRM(param_no, data_type, int_val, real_val, str_val, status)
   IF ( status <> 0 ) THEN     -- if parameter is not existed, post the error.
```

```
    POST_ERR(status,'',0,0)
  ELSE
    IF ( data_type = TYPE_STRING) THEN -- parameter is STRING type
      --get the selected device information
      GET_VAR(entry,'*SYSTEM*', '$DEVICE', dev_name, status)
      IF ( status <> 0) THEN
        POST_ERR( status, '', 0, 0)
      ELSE
        list_name = dev_name + '¥' + str_val + '.dt'
      ENDIF

      OPEN FILE filelist ('RO',list_name)
      status = IO_STATUS(filelist) -- Get status of OPEN FILE
      IF status = 0 THEN
        --Clear user screen to display LOAD status
        WRITE TPDISPLAY (CHR(128),CHR(137))
        REPEAT
          READ filelist(oneprog) -- Read one line from specified file list
          status = IO_STATUS(filelist) -- Get status of READ
          IF status = 0 THEN -- program is found
            -- copy the listed program to the controller
            COPY_FILE(dev_name + '¥' + oneprog, 'MD:' + oneprog, TRUE,
 FALSE, status)
            IF (status <> 0 ) THEN    -- if copy was failed, post the error.
              POST_ERR( status, '', 0, 0)
              IF (status = 2014) THEN -- FILE -014 File not found
                -- Force the USER menu screen
                FORCE_SPMENU(TP_PANEL,SPI_TPUSER,1)
                -- display the file name that is not found
                WRITE TPDISPLAY (oneprog, ' is not found',CR)
              ENDIF
            ENDIF
          ELSE
            IF (status <> 2021) THEN
              POST_ERR( status, '', 0, 0)
            ENDIF
          ENDIF
        UNTIL status = 2021 -- Repeat until end of file
        CLOSE FILE filelist
      ELSE
        POST_ERR( status, '', 0, 0)
        IF (status = 2014) THEN -- FILE -014 File not found
          -- Force the USER menu screen
          FORCE_SPMENU(TP_PANEL,SPI_TPUSER,1)
          --Clear user screen to display copy
          WRITE TPDISPLAY (CHR(128),CHR(137))
          WRITE TPDISPLAY (list_name, ' is not found',CR)
        ENDIF
      ENDIF
    ELSE
```

```
        FORCE_SPMENU(TP_PANEL,SPI_TPUSER,1) -- Force the USER menu
     screen
        WRITE TPDISPLAY (CHR(128),CHR(137))
        WRITE TPDISPLAY ('Wrong file list name was input.',CR)
        WRITE TPDISPLAY ('Please input correct file list.', CR,CR)
        WRITE TPDISPLAY ('LOAD usage:',CR)
        WRITE TPDISPLAY ('CALL LOAD(file_list)',CR)
      ENDIF
   ENDIF

END load
```

＜save.kl＞

```
PROGRAM save

---------------------------------------------------------------------
-- This program save is to save the TP program from specified
-- file list.
-- Usage:
-- Call save(filelist)
---------------------------------------------------------------------

%NOLOCKGROUP
%NOPAUSE   = ERROR + COMMAND + TPENABLE
%NOABORT = ERROR + COMMAND + TPENABLE
%NOBUSYLAMP

CONST
   TYPE_STRING = 3

VAR
   status        : INTEGER
   param_no     : INTEGER
   data_type    : INTEGER
   int_val       : INTEGER
   real_val     : REAL
   str_val       : STRING[50]
   filelist     : FILE
   entry         : INTEGER
   dev_name     : STRING[10]
   list_name    : STRING[60]
   oneprog       : STRING[40]

---------------------------------------------------------------------
BEGIN
---------------------------------------------------------------------
   status = 0
   str_val = ''
   dev_name = ''
   list_name = ''
```

```
oneprog = ''
param_no = 1

-- get parameter 1 that is the list of file to be loaded.
GET_TPE_PRM(param_no, data_type, int_val, real_val, str_val, status)
IF ( status <> 0 ) THEN      -- if parameter is not existed, post the error.
   POST_ERR(status,'',0,0)
ELSE
   IF ( data_type = TYPE_STRING) THEN -- parameter is STRING type
      --get the selected device information
      GET_VAR(entry,'*SYSTEM*', '$DEVICE', dev_name, status)
      IF ( status <> 0) THEN
         POST_ERR( status, '', 0, 0)
      ELSE
         list_name = dev_name + '¥' + str_val + '.dt'
      ENDIF

      OPEN FILE filelist ('RO',list_name)
      status = IO_STATUS(filelist) -- Get status of OPEN FILE
      IF status = 0 THEN
         --Clear user screen to display SAVE status
         WRITE TPDISPLAY (CHR(128),CHR(137))
         REPEAT
            READ filelist(oneprog) -- Read one line from specified file list
            status = IO_STATUS(filelist) -- Get status of READ
            IF status = 0 THEN -- program is found
               -- copy the listed program to the controller
               COPY_FILE('MD:' + oneprog, dev_name + '¥' + oneprog, TRUE,
 FALSE, status)
               IF (status <> 0 ) THEN    -- if copy was failed, post the error.
                  POST_ERR( status, '', 0, 0)
                  IF (status = 2014) THEN -- FILE-014 Program does not exist
                     -- Force the USER menu screen
                     FORCE_SPMENU(TP_PANEL,SPI_TPUSER,1)
                     -- display the file name that is not found
                     WRITE TPDISPLAY (oneprog, ' does not exist',CR)
                  ELSE
                     IF (status = 7073) THEN -- MEMO-073 Program does not exist
                        -- Force the USER menu screen
                        FORCE_SPMENU(TP_PANEL,SPI_TPUSER,1)
                        -- display the file name that is not found
                        WRITE TPDISPLAY (oneprog, ' does not exist',CR)
                     ENDIF
                  ENDIF
               ENDIF
            ELSE
               IF (status <> 2021) THEN
                  POST_ERR( status, '', 0, 0)
               ENDIF
            ENDIF
```

```
            UNTIL status = 2021 -- Repeat until end of file
            CLOSE FILE filelist
          ELSE
            POST_ERR( status, '', 0, 0)
            IF (status = 2014) THEN -- FILE -014 File not found
              -- Force the USER menu screen
              FORCE_SPMENU(TP_PANEL,SPI_TPUSER,1)
              --Clear user screen to display copy
              WRITE TPDISPLAY (CHR(128),CHR(137))
              WRITE TPDISPLAY (list_name, ' is not found',CR)
            ENDIF
          ENDIF
        ELSE
          FORCE_SPMENU(TP_PANEL,SPI_TPUSER,1) -- Force the USER menu
      screen
          WRITE TPDISPLAY (CHR(128),CHR(137))
          WRITE TPDISPLAY ('Wrong file list name was input.',CR)
          WRITE TPDISPLAY ('Please input correct file list.', CR,CR)
          WRITE TPDISPLAY ('SAVE usage:',CR)
          WRITE TPDISPLAY ('CALL SAVE(file_list)',CR)
        ENDIF
    ENDIF

END save
```

＜delete.kl＞

```
PROGRAM delete

--------------------------------------------------------------------------
-- This program delete is to delete the TP program from specified
-- file list.
-- Usage:
-- Call delete(filelist)
--------------------------------------------------------------------------

%NOLOCKGROUP
%NOPAUSE   = ERROR + COMMAND + TPENABLE
%NOABORT = ERROR + COMMAND + TPENABLE
%NOBUSYLAMP

CONST
  TYPE_STRING = 3

VAR
  status       : INTEGER
  param_no     : INTEGER
  data_type    : INTEGER
  int_val      : INTEGER
  real_val     : REAL
  str_val      : STRING[50]
```

```
    filelist     : FILE
    entry        : INTEGER
    dev_name     : STRING[10]
    list_name    : STRING[60]
    oneprog      : STRING[40]

-------------------------------------------------------------------------
BEGIN
-------------------------------------------------------------------------
    status = 0
    str_val = ''
    dev_name = ''
    list_name = ''
    oneprog = ''
    param_no = 1

    -- get parameter 1 that is the list of file to be loaded.
    GET_TPE_PRM(param_no, data_type, int_val, real_val, str_val, status)
    IF ( status <> 0 ) THEN     -- if parameter is not existed, post the error.
       POST_ERR(status,'',0,0)
    ELSE
       IF ( data_type = TYPE_STRING) THEN    -- parameter is STRING type
          --get the selected device information
          GET_VAR(entry,'*SYSTEM*', '$DEVICE', dev_name, status)
          IF ( status <> 0) THEN
             POST_ERR( status, '', 0, 0)
          ELSE
             list_name = dev_name + '¥' + str_val + '.dt'
          ENDIF

          OPEN FILE filelist ('RO',list_name)
          status = IO_STATUS(filelist) -- Get status of OPEN FILE
          IF status = 0 THEN
             --Clear user screen to display DELETE status
             WRITE TPDISPLAY (CHR(128),CHR(137))
             REPEAT
                READ filelist(oneprog) -- Read one line from specified file list
                status = IO_STATUS(filelist) -- Get status of READ
                IF status = 0 THEN -- program is found
                   -- copy the listed program to the controller
                   CLEAR(oneprog, status)
                   IF (status <> 0 ) THEN    -- if copy was failed, post the error.
                      POST_ERR( status, '', 0, 0)
                      IF (status = 7073) THEN -- MEMO -073 File not found
                         -- Force the USER menu screen
                         FORCE_SPMENU(TP_PANEL,SPI_TPUSER,1)
                         -- display the file name that is not found
                         WRITE TPDISPLAY (oneprog, ' does not exist',CR)
                      ENDIF
                   ENDIF
```

```
            ELSE
              IF (status <> 2021) THEN
                POST_ERR( status, '', 0, 0)
              ENDIF
            ENDIF
          UNTIL status = 2021 -- Repeat until end of file
          CLOSE FILE filelist
        ELSE
          POST_ERR( status, '', 0, 0)
          IF (status = 2014) THEN -- FILE -014 File not found
            -- Force the USER menu screen
            FORCE_SPMENU(TP_PANEL,SPI_TPUSER,1)
            --Clear user screen to display copy
            WRITE TPDISPLAY (CHR(128),CHR(137))
            WRITE TPDISPLAY (list_name, ' is not found',CR)
          ENDIF
        ENDIF
      ELSE
        FORCE_SPMENU(TP_PANEL,SPI_TPUSER,1) -- Force the USER menu
    screen
        WRITE TPDISPLAY (CHR(128),CHR(137))
        WRITE TPDISPLAY ('Wrong file list name was input.',CR)
        WRITE TPDISPLAY ('Please input correct file list.', CR,CR)
        WRITE TPDISPLAY ('DELETE usage:',CR)
        WRITE TPDISPLAY ('CALL DELETE(file_list)',CR)
      ENDIF
    ENDIF

END delete
```

5 CREATING SCREEN WITH FORM EDITOR

Form Editor is the software to display screen and handle key input. This chapter explains creating screen with Form Editor.

5.1 OVERVIEW

Below is the step to create screens
Creating step:

1. Create dictionary file whose extension is ftx
2. Create TX, dictionary file, and VR, variable file, files,
3. Create KAREL program and translate it
4. Verify screen on ROBOGUIDE
5. Load it to the robot controller.
6. Verify screen on the robot

See chapter 10 of KAREL Reference Manual to know more information. Systematic explication is written in it. This section is explained to create easy screen with ROBOGUIDE.

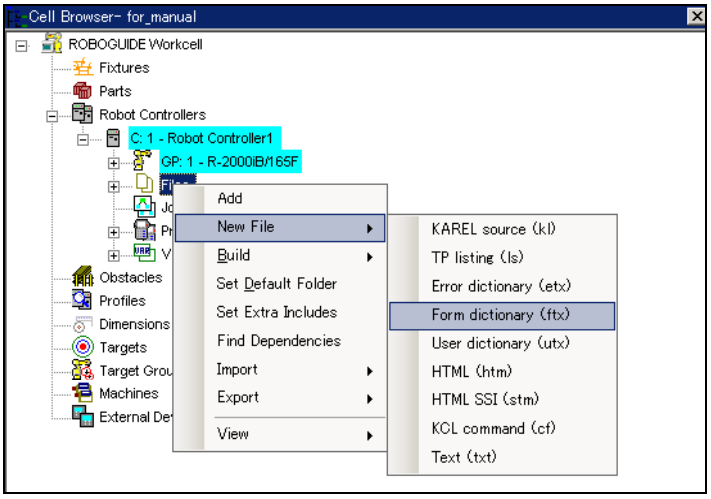
5.2 CREATE DICTIONARY FILES

5.2.1 Dictionary Files

Dictionary file has text to display and data to control screen. It allows KAREL program to show the content of dictionary files. Because the program and displaying data are separated, it is easy to fix the screen without program fixing. In some case it needs to fix the program itself. In case that the robot controller applies multi languages such as Japanese and English, it is needed to switch languages depending on them. If you prepare each language dictionary file, KAREL program automatically use its file respond to now-selected language. It is advantage of dictionary file.

5.2.2 Create Dictionary Whose Extension is FTX

See the help of ROBOGUIDE “Development tool” for dictionary creating. Do right click on cell browser, select “new file”. Select “Add” if you would like to add the created dictionary. In the case of newly creating, select “Form Dictionary”. Following is the sample. (If cell browser is not displayed, select from menu.)



Adding newly file is empty. It is named like “Untitled1.ftx”. Click “Save As” button and input dictionary name according to rule that first 4 characters are dictionary name and next 4 characters are language name. For example, about “dictengl.ftx”, “dict” is dictionary name and “engl” is language name.

Three files, flstengl.ftx, flstjapa.ftx, flstkanj.ftx, are prepared here for robot of multi languages. Each dictionary is English, Japanese (KANA: for legacy pendant) and Japanese (KANJI: for *i*Pendant).
ftstengl.ftx (for English)

```
.kl ftstex
.form
$-,form1
&home &reverse "Sample screen" &standard &new_line
" Sample label " &new_line
@3,5" Integer: "     @3,28"-%10d(1,32767)" &new_line
@4,5" Real: "            @4,28"-%12f" &new_line
@5,5" Program name(TP):"     @5,28 "-%12pk(1)" &new_line
@6,5" Program name(PC): "    @6,28"-%12pk(2)" &new_line
@7,5" DIN[1]               " @7,28"-%7P(io_c)" &new_line
@8,5" DOUT[1]          " @8,28"-%7P(io_c)" &new_line
@9,5" Select item"    @9,28"-%12w(item_c)" &new_line
^form1_fkey * specifies element which contains
* function key labels
?form1_help * element which contains help
.endform
$-,form1_fkey * function key labels
" F1" &new_line
" F2" &new_line
" F3" &new_line
" F4" &new_line
" HELP"
$-, form1_help * help text
"Help Line 1" &new_line
"Help Line 2" &new_line
"Help Line 3" &new_line
* You can have a maximum of 48 help lines
$-, io_c
" OFF" &new_line
" ON "
$-, item_c
"item 1"
$-
```

```
"item 2"
$-
"item 3"
$-
"¥a"
```

ftstjapa.ftx (for Legacy pendant)

```
.form
$-,form1
&home &reverse "ｻﾝﾌﾟﾙ ｶﾞﾒﾝ" &standard &new_line
"ｻﾝﾌﾟﾙ ﾗﾍﾞﾙ " &new_line
@3,5" Integer: "     @3,28"-%10d(1,32767)" &new_line
@4,5" Real: "          @4,28"-%12f" &new_line
@5,5" ﾌﾟﾛｸﾞﾗﾑ(TP):"     @5,28 "-%12pk(1)" &new_line
@6,5" ﾌﾟﾛｸﾞﾗﾑ(PC): "   @6,28"-%12pk(2)" &new_line
@7,5" DIN[1]              " @7,28"-%7P(io_c)" &new_line
@8,5" DOUT[1]         " @8,28"-%7P(io_c)" &new_line
@9,5" Select item"   @9,28"-%12w(item_c)" &new_line
^form1_fkey * specifies element which contains
* function key labels
?form1_help * element which contains help
.endform
$-,form1_fkey * function key labels
" F1" &new_line
" F2" &new_line
" F3" &new_line
" F4" &new_line
" HELP"
$-, form1_help * help text
"ﾍﾙﾌﾟ 1" &new_line
"ﾍﾙﾌﾟ 2" &new_line
"ﾍﾙﾌﾟ 3" &new_line
* You can have a maximum of 48 help lines
$-, io_c
" ｵﾌ " &new_line
" ｵﾝ "
$-, item_c
"item 1"
$-
"item 2"
$-
"item 3"
$-
"¥a"
```

ftstkanj.ftx (for *i*Pendant)

```
.form
$-,form1
&home &reverse 'サンプル画面' &standard &new_line
'サンプルラベル' &new_line
@3,5" Integer: "     @3,23"-%10d(1,32767)" &new_line
@4,5" Real: "          @4,23"-%12f" &new_line
@5,5' プログラム(TP):'     @5,23 "-%12pk(1)" &new_line
@6,5' プログラム(PC): '   @6,23"-%12pk(2)" &new_line
@7,5" DIN[1]              " @7,23"-%7P(io_c)" &new_line
@8,5" DOUT[1]          " @8,23"-%7P(io_c)" &new_line
@9,5'項目選択'   @9,23"-%12w(item_c)" &new_line
^form1_fkey * specifies element which contains
* function key labels
?form1_help * element which contains help
.endform
$-,form1_fkey * function key labels
" F1" &new_line
" F2" &new_line
" F3" &new_line
" F4" &new_line
" HELP"
$-, form1_help * help text
"ﾍﾙﾌﾟ 1" &new_line
"ﾍﾙﾌﾟ 2" &new_line
"ﾍﾙﾌﾟ 3" &new_line
* You can have a maximum of 48 help lines
$-, io_c
" ｵﾌ " &new_line
" ｵﾝ "
$-, item_c
'項目 1'
$-
'項目 2'
$-
'項目 3'
$-
"¥a"
```

5.2.3 Dictionary Name and Languages

Dictionary name has following rules.

- File name is 8 characters.
- First 4 characters are dictionary name.
- Last 4 characters are language name.
- The extension is one of etx, ftx or utx.
- Ftx is to use Form Editor.

- Following is language name of 4 characters.
  "ENGL" = ENGLISH
  "JAPA" = JAPANESE
  "KANJ" = KANJI
  "FREN" = FRENCH
  "GERM" = GERMAN
  "SPAN" = SPANISH
  "CHIN" = CHINESE
  "TAIW" = TAIWANESE

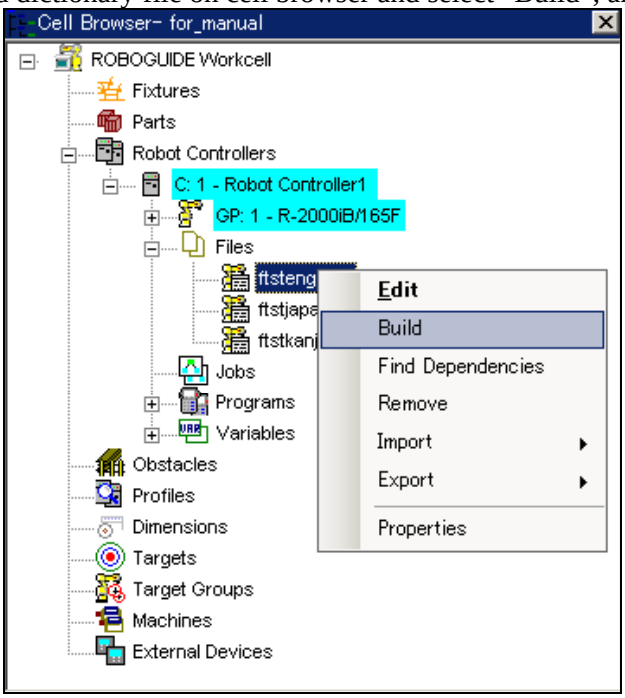
In this case, three files are created.

- ftstengl.ftx
- ftstjapa.ftx
- ftstkanj.ftx

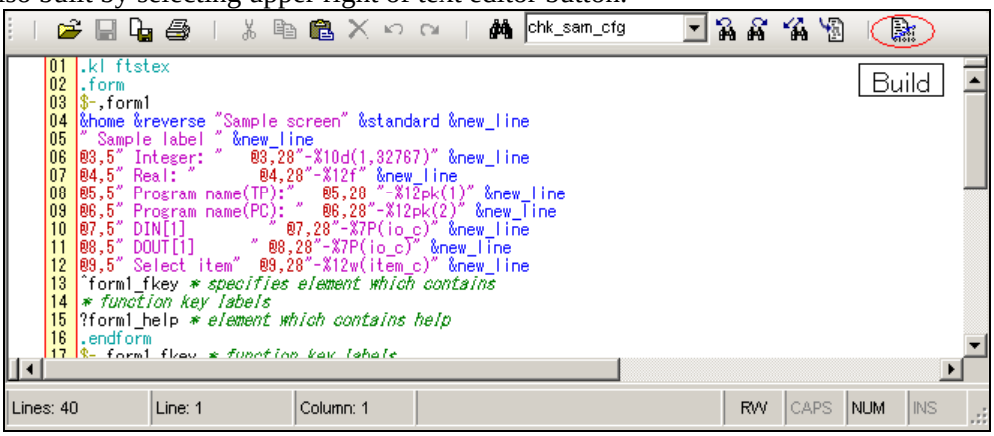
Dictionary name, “flst”, is common independent on languages. The language name is engl, japa and kanj in above case.

5.2.4 Dictionary File Build

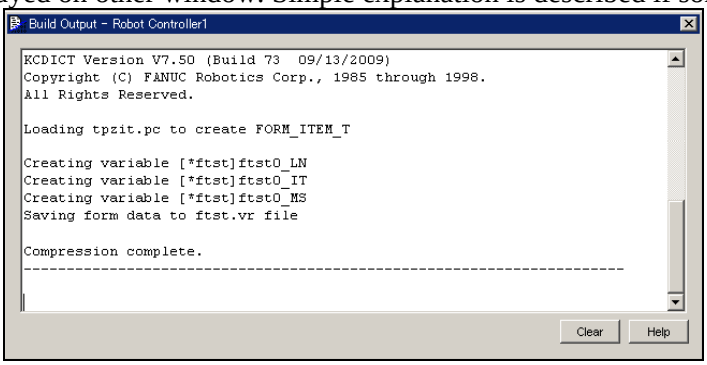
Do right click on the added dictionary file on cell browser and select “Build”, and then it is built.



They are also built by selecting upper right of text editor button.



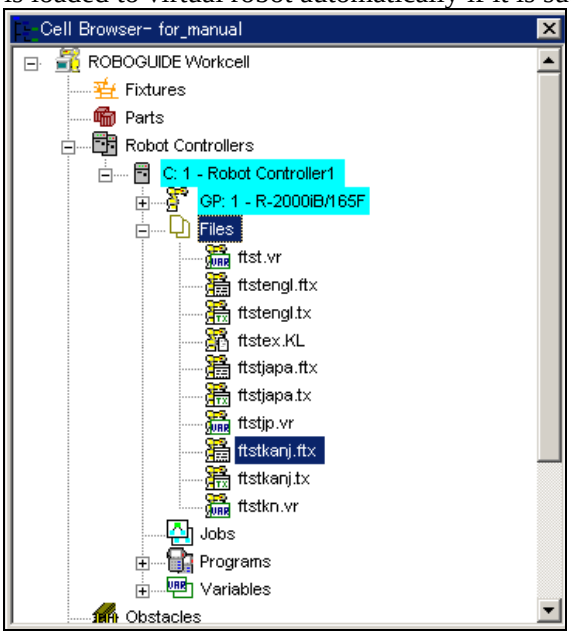
Result of build is displayed on other window. Simple explanation is described if some errors are occurred



Following files are created.

- For loading:
  ftstengl.tx, ftstjapa.tx, ftstkanj,tx
- Variable files
  ftst.vr, ftstjp.vr, ftstkn.vr
- KAREL file for including
  ftstex.kl

Each file is created in the same directory of built file. See KAREL Reference Manual for detail.
Translated KAREL program is loaded to virtual robot automatically if it is successfully built.



5.3 CREATE KAREL PROGRAMS

Create below formtest.kl and translate (It is the same way of dicitionary file to create and translae).

```
PROGRAM formtest
%INCLUDE klevkmsk
%INCLUDE klevkeys
%INCLUDE ftstex
VAR
   l_status:INTEGER
   value_array: ARRAY[7] OF STRING[30]
   change_array:ARRAY[1] OF BOOLEAN --Allows setting size [1] if not use
   inact_array:ARRAY[1] OF BOOLEAN--Allows setting size [1] if not use
   test_int:INTEGER
   test_real:REAL
   prog_name1:STRING[40]
   prog_name2:STRING[40]
   select_item: INTEGER
   def_item :INTEGER
   term_mask:INTEGER
   term_char:INTEGER
   exit_menu:BOOLEAN
   device_stat:INTEGER
BEGIN
   test_int    = 12345
   test_real = 12.345
   -- There are 7 data to display such as %d in *.ftx
   -- Then value_array is ARRAY[7]
   value_array[1] = 'test_int'    --First data item %10d is test_int of INTEGER
   value_array[2] = 'test_real' --Second %12f is test_real of REAL
   value_array[3] = 'prog_name1'--Third %12pk is prog_name1 of STRING
   value_array[4] = 'prog_name2'--Fourth %12pk is prog_name2 of STRING
   value_array[5] = 'DIN[1]'       --Fifth %7P is BOOLEAN to show DI[1]
   value_array[6] = 'DOUT[1]'      --Sixth %7P is BOOLEAN to show DO[1]
   value_array[7] = 'select_item' --Seventh %12w is INTEGER

   def_item = 1
   term_mask = kc_func_key
   --Force to switch USER2 screen
   FORCE_SPMENU(device_stat, SPI_TPUSER2, 1)
   exit_menu = FALSE
   REPEAT
      --Display and wait for input keys.
      DISCTRL_FORM('ftst', form1, value_array, inact_array, change_array,
                   term_mask, def_item, term_char, l_status)
      IF term_char = KY_NEW_MENU THEN
         --Exit loop if requested.
         exit_menu = TRUE
      else

      ENDIF
```

```
   UNTIL exit_menu
END formtest
```

The DISCTRL_FORM built-in is used to display and control a form on the teach pendant or CRT/KB screens.

Syntax : DISCTRL_FORM(dict_name, ele_number, value_array, inact_array, change_array, term_mask, def_item, term_char, status)

Input/Output Parameters :
[in] dict_name : STRING
[in] ele_number : INTEGER
[in] value_array : ARRAY OF STRING
[in] inactive_array : ARRAY OF BOOLEAN
[out] change_array : ARRAY OF BOOLEAN
[in] term_mask : INTEGER
[in,out] def_item : INTEGER
[out] term_char : INTEGER
[out] status : INTEGER
%ENVIRONMENT Group :PBcore

Details:

- dict_name is the four-character name of the dictionary containing the form.
- ele_number is the element number of the form.
- value_array is an array of variable names that corresponds to each edit or display only data item in the form. Each variable name can be specified as a '[prog_name]var_name'.
  - [prog_name] is the name of the program that contains the specified variable. If [prog_name] is not specified, the current program being executed is used. '[*SYSTEM*]' should be used for system variables.
  - var_name must refer to a static, global program variable.
  - var_name can contain node numbers, field names, and/or subscripts.
  - var_name can also specify a port variable with index. For example, 'DIN[1]'.
- inactive_array is an array of booleans that corresponds to each item in the form.
  - Each boolean defaults to FALSE, indicating it is active.
  - You can set any boolean to TRUE which will make that item inactive and non-selectable.
  - The array size can be greater than or less than the number of items in the form.
  - If an inactive_array is not used, then an array size of 1 can be used. The array does not need to be initialized.
- change_array is an array of booleans that corresponds to each edit or display only data item in the form.
  - If the corresponding value is set, then the boolean will be set to TRUE, otherwise it is set to FALSE. You do not need to initialize the array.
  - The array size can be greater than or less than the number of data items in the form.
  - If change_array is not used, an array size of 1 can be used.
- term_mask is a bit-wise mask indicating conditions that will terminate the form.
- DISCTRL_FORM will display the form on the teach pendant device. To display the form on the CRT/KB device, you must create an INTEGER variable, device_stat , and set it to crt_panel . To set control to the teach pendant device, set device_stat to tp_panel.
- status explains the status of the attempted operation. If status returns a value other than 0, an error has occurred.

FORCE_SPMENU Built-In Procedure

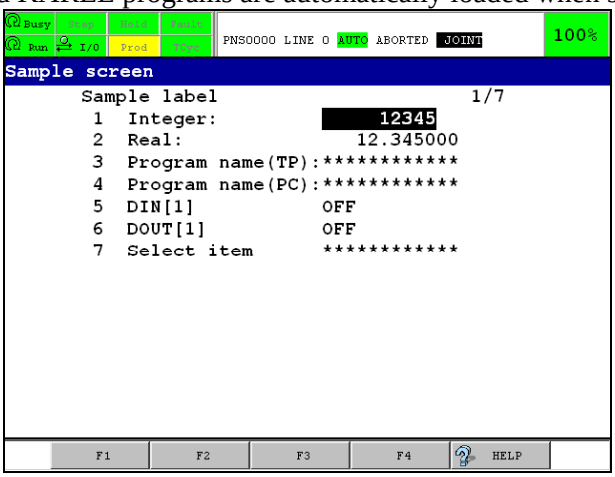
Purpose: Forces the display of the specified menu
Syntax : FORCE_SPMENU(device_code, spmenu_id, screen_no)

Input/Output Parameters :
[in] device_code :INTEGER
[in] spmenu_id :INTEGER
[in] screen_no :INTEGER
%ENVIRONMENT Group :pbcore

Details:
- device_code specifies the device and should be one of the following predefined constants:
- tp_panel Teach pendant device
- crt_panel CRT device
- spmenu_id and screen_no specify the menu to force. See KAREL Reference Manual, Appendix A.

5.4 CONFIRM ON ROBOGUIDE

Execute KAREL program, FORMTEST, on the virtual robot. KAREL program is allowed only executing because dictionary files and KAREL programs are automatically loaded when successfully built. *i*Pendant



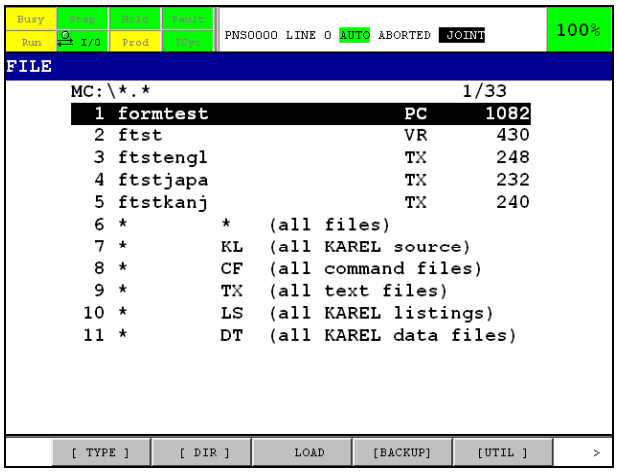
When you set the cursor onto “3 Program Name”, [CHOICE] is displayed on F4 key. Push F4, then TP program list is displayed. Select one of them, then it is displayed on the screen. Also, in case of “4 Program Name”, KAREL program list is displayed after pushing F4 key

Set the cursor onto DOUT[1], then F4[ON] and F4[OFF] are displayed. Pushing F4 or F5 allows DO[1] to switch ON or OFF.
Set the cursor onto “7 Select Item”, then F4 key turns to F4[CHOICE]. Defined item_c in ftstengl.ftx are displayed. In this sample item 1~ 3 are displayed.

Pushing F5[HELP] displays the text of form1_help in ftstengl.ftx. In this sample following is displayed.
Help Line 1
Help Line 2
Help Line 3

5.5 LOAD TO THE ROBOT CONTROLLER

After confirming on virtual robot, next is to confirm on actual robot.
1. Copy PC, VR, TX files to Compact FLASH ATA card (called MC: hereinafter).
2. Insert the MC: to the robot controller.
3. Display FILE screen. It allows PC, VR, TX files to load with F3[LOAD]



WARNING

Only FTST.VR is used because VR file includes starting position and width of data. VR file is created each language but enabled data is only one of them. Then you must unify kind, position, width of data in each language. Only one VR file is loaded. Usually, we recommend English VR file to use.

5.6 CONFIRM ON ACTUAL ROBOT

Execute foromtest.pc on the robot controller.

6 EDIT WITH FORM EDITOR

This chapter explains about editing with Form Editor based on created screen in previous chapter. Below description is the sample as selected language is English. Only ftstengl.ftx is used to explain but you should change ftstejapa.ftx and ftstkanj.ftx.
See KAREL Reference Manual to know detail of Form Editor.

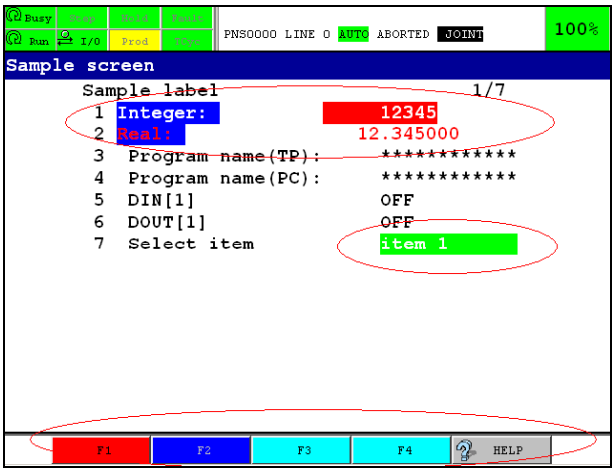
6.1 COLOR CHANGE

Change ftstengl.ftx as follows. Also you must change ftstjapa.ftx and ftstkanj.ftx. Build it and confirm on ROBOGUIDE. (Execute formtest.pc. No need to change formtest.kl)

```
.kl ftstex
.form
$-,form1
&home &reverse "Sample screen" &standard &new_line
" Sample label " &new_line
*Left side item: background set blue, figure is white.
*Right side: figure is red, background is default.
@3,5&fg_white &bg_blue "Integer: "&bg_dflt &fg_red    @3,23"-%10d(1,32767)"
    &new_line
*Figure stays red, left side background is set blue.
&bg_blue@4,5"Real: "     &bg_dflt       @4,23"-%12f" &fg_dflt &new_line
@5,5" Program name(TP):"     @5,28 "-%12pk(1)" &new_line
@6,5" Program name(PC): "    @6,28"-%12pk(2)" &new_line
@7,5" DIN[1]                " @7,28"-%7P(io_c)" &new_line
@8,5" DOUT[1]           " @8,28"-%7P(io_c)" &new_line
@9,5" Select item"    @9,28"-%12w(item_c)" &new_line
^form1_fkey * specifies element which contains
* function key labels
?form1_help * element which contains help
.endform
$-,form1_fkey * function key labels
&bg_red" F1" &new_line    *Background is red
&bg_blue &fg_white" F2"&fg_dflt    &new_line    *Background is blue, figure is white
&bg_cyan" F3" &new_line *Back ground is cyan
" F4" &new_line *No color is specified in this line. Then background stays cyan with
    F3
&bg_dflt " HELP" *Set background to default
$-, form1_help * help text
"Help Line 1" &new_line
"Help Line 2" &new_line
"Help Line 3" &new_line
* You can have a maximum of 48 help lines
$-, io_c
" OFF" &new_line
" ON "
$-, item_c
&fg_white &bg_green "item 1"    *item 1, green
```

```
 $-
 "item 2"
 $-
 "item 3"
 $-
"¥a"
```

Background and character colors are changed as follows. In “Select Item”, only the case of selecting “Item 1” is shown green.



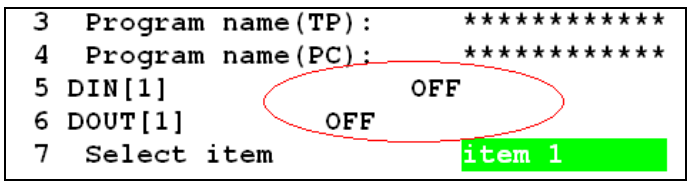
In ftx file, &bg_** specifies a color of background. Specified color is shown until other color are specified or &bg_dflt is described.
&fg_** specifies a character color. As with &bg_**, specified color is shown until other color are specified or &fg_dflt is described.
See KAREL Reference Manual if you know enable color to specify .

6.2 ARRANGEMENT CHANGE

Using “@” specifies where dictionary elements put in. For example, parts of DIN[1] and DOUT[1] in fstsengl.ftx positions are changed @7,23->@7,25 and @8,23->@8,20.

Arrangement specified like “@height, width”.
@7,5"DIN[1]" @7,25"-%7P(io_c)" &new_line
@8,5"DOUT[1]" @8,20"-%7P(io_c)" &new_line

Confirm the screen on ROBOGUIDE after building the dictionary.



Change the arrangement to previous (@7,25->@7,23, @8,20->@8,23) and proceed next.

6.3 DISABLE TO EDIT CONFIG

Change “-%7P(io_c)” to “%7P(io_c)” in line of “@8,5"DOUT[1]" @8,23"-%7P(io_c)" &new_line” in fstsengl.ftx.

Confirm the screen on ROBOGUIDE after building the dictionary.

```
5   DIN[1]                OFF
    DOUT[1]               OFF
6   Select item           item 1
```

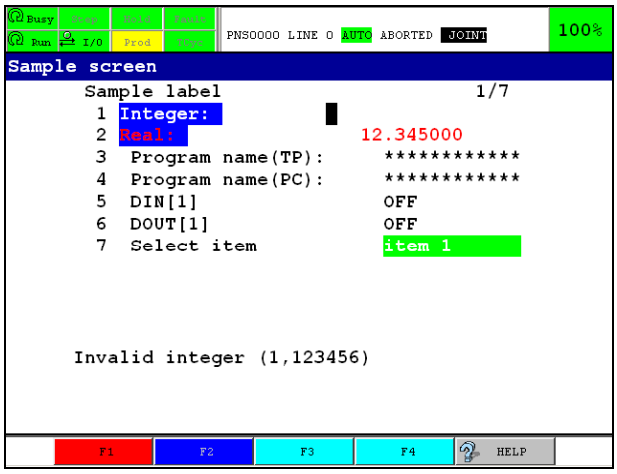
It does not allow the cursor to set on DOUT[1], then it cannot edit. In ftx file, “-“ means it can edit. (Another way to disable editing is to set inactive_array to TRUE in DISCTRL_FORM built-in function.) Return the config to “-%7P(io_c)” and proceed next.

6.4 RESTRICT INPUT VALUE

Under “Sample label” of ftstengl.ftx, it says
“@3,5&fg_white &bg_blue "Integer: "&bg_dflt &fg_red @3,23"-%10d(1,32767)" &new_line”.
Now explaining about @3,23"-%10d(1,32767)" meaning. @3,23 is specified the position of data, line 3, row 23. "-%10d(1,32767)" means that the range of shown value. “-“ means that it allows editing.
The “d” of %10d means it is decimal number. 10 means precision, then it shows 10 digits integer.
(1,32767) shows the range to input. It allows integer from 1 to 32767 to input.

Try to change like @3,23"-%10d (1,123456)”. Confirm the screen on ROBOGUIDE after building the dictionary. (Execute formtest.pc.)

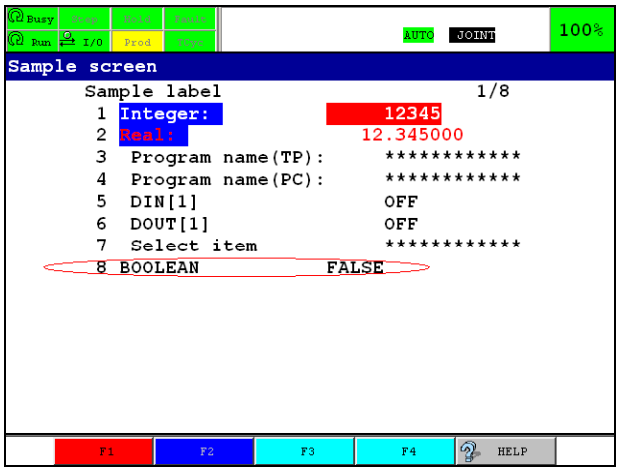
Set the cursor onto “Integer” and input out of range value such as 0. It is not allowed to input.



Input a value and confirm that it is displayed normally.

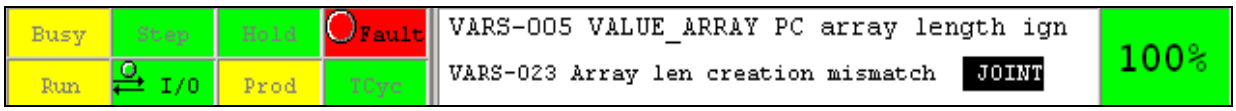
6.5 ADD DISPLAY ELEMENTS

This section explains the way to add display elements as follows. To add display elements it needs to change not dictionary file (ex. fstsengl.ftx) but KAREL program file (ex. formtest.kl).

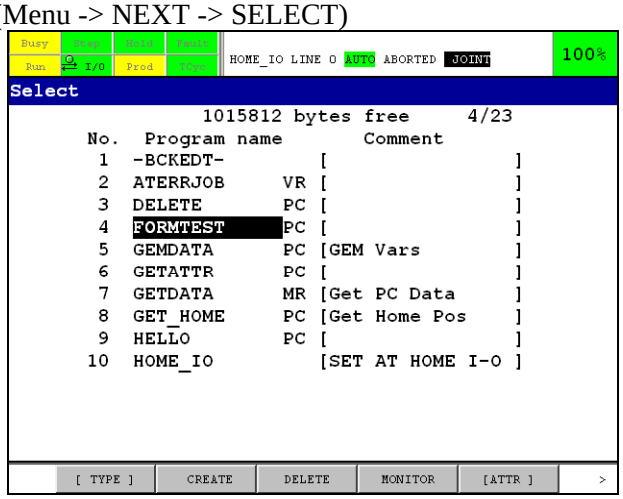


In the case of changing array of KAREL variables, you must delete vr file that it has old variables to be add. It may not work normally.

If “VARS-005 ** PC array length ignored. VARS-023 Array len creation mismathch” is posted on upper right of teach pendant, it may not enable the change.



Delete corresponding PC and VR files as follows.
Display SELECT screen. (Menu -> NEXT -> SELECT)



Set the cursor onto formtest.pc and F3[DELETE]. Confirming message is displayed. Then select F4[YES].
After deleting formtest.pc, formtest.vr is shown soon. Delete its VR file too.

Change formtest.kl as follows.

```
PROGRAM formtest
%INCLUDE klevkmsk
%INCLUDE klevkeys
%INCLUDE ftstex
VAR
   l_status:INTEGER
   value_array: ARRAY[8] OF STRING[30] --Change to 8 from 7
   change_array:ARRAY[1] OF BOOLEAN --Allows setting size [1] if not use
```

```
    inact_array:ARRAY[1] OF BOOLEAN--Allows setting size [1] if not use
    test_int:INTEGER
    test_real:REAL
    prog_name1:STRING[40]
    prog_name2:STRING[40]
    select_item: INTEGER
    def_item :INTEGER
    term_mask:INTEGER
    term_char:INTEGER
    exit_menu:BOOLEAN
    device_stat:INTEGER
  BEGIN
    test_int    = 12345
    test_real = 12.345
    -- There are 7 data to display such as %d in *.ftx
    -- Then value_array is ARRAY[7]
    value_array[1] = 'test_int'    --First data item %10d is test_int of INTEGER
    value_array[2] = 'test_real' --Second %12f is test_real of REAL
    value_array[3] = 'prog_name1'--Third %12pk is prog_name1 of STRING
    value_array[4] = 'prog_name2'--Fourth %12pk is prog_name2 of STRING
    value_array[5] = 'DIN[1]'       --Fifth %7P is BOOLEAN to show DI[1]
    value_array[6] = 'DOUT[1]'     --Sixth %7P is BOOLEAN to show DO[1]
    value_array[7] = 'select_item' --Seventh %12w is INTEGER
    value_array[8] = 'bl_sample' --Eighth %7P is BOOLEAN

    def_item = 1
    term_mask = kc_func_key
    --Force to switch USER2 screen
    FORCE_SPMENU(device_stat, SPI_TPUSER2, 1)
    exit_menu = FALSE
    REPEAT
      --Display and wait for input keys.
      DISCTRL_FORM('ftst', form1, value_array, inact_array, change_array,
                   term_mask, def_item, term_char, l_status)
      IF term_char = KY_NEW_MENU THEN
        --Exit loop if requested.
        exit_menu = TRUE
      else

      ENDIF
    UNTIL exit_menu
  END formtest
```

Change value_array[7] to value_array[8]. Add a variable of bl_sample.
Change ftstengl.fix as follows. (Along with fstsjapa.ftx and ftstkanj.ftx)

```
.kl ftstex
.form
$-,form1
&home &reverse "Sample screen" &standard &new_line
```

```
" Sample label " &new_line
*Left side item: background set blue, figure is white.
*Right side: figure is red, background is default.
@3,5&fg_white &bg_blue "Integer: "&bg_dflt &fg_red    @3,23"-%10d(1,32767)"
    &new_line
*Figure stays red, left side background is set blue.
&bg_blue@4,5"Real: "     &bg_dflt       @4,23"-%12f" &fg_dflt &new_line
@5,5" Program name(TP):"     @5,28 "-%12pk(1)" &new_line
@6,5" Program name(PC): "    @6,28"-%12pk(2)" &new_line
@7,5" DIN[1]                " @7,28"-%7P(io_c)" &new_line
@8,5" DOUT[1]          " @8,28"-%7P(io_c)" &new_line
@9,5" Select item"    @9,28"-%12w(item_c)" &new_line
@10,5 "BOOLEAN"@10,23"-%7B(bool_c)" &new_line
^form1_fkey * specifies element which contains
* function key labels
?form1_help * element which contains help
.endform
$-,form1_fkey * function key labels
&bg_red" F1" &new_line    *Background is red
&bg_blue &fg_white" F2"&fg_dflt    &new_line    *Background is blue, figure is white
&bg_cyan" F3" &new_line *Back ground is cyan
" F4" &new_line *No color is specified in this line. Then background stays cyan with
    F3
&bg_dflt " HELP" *Set background to default
$-, form1_help * help text
"Help Line 1" &new_line
"Help Line 2" &new_line
"Help Line 3" &new_line
* You can have a maximum of 48 help lines
$-, io_c
" OFF" &new_line
" ON "
$-, item_c
&fg_white &bg_green "item 1"    *item 1, green
$-
"item 2"
$-
"item 3"
$-
"¥a"

$-,bool_c
"FALSE" &new_line
"TRUE"
```

Add following

```
@10,5 "BOOLEAN"@10,23"-%7B(bool_c)" &new_line
$-,bool_c
"FALSE" &new_line
"TRUE"
```

One byte character is covered “ (double quotation), two bytes character is covered ‘ (single quotation). In “engl” and “japa” dictionaries, it is not allowed to use two bytes characters.

Ex: @10,5 “BOOLEAN TYPE”@10,23"-%7B(bool_c)" &new_line
Confirm the screen on ROBOGUIDE after building and translating the dictionary. New item is added as above.

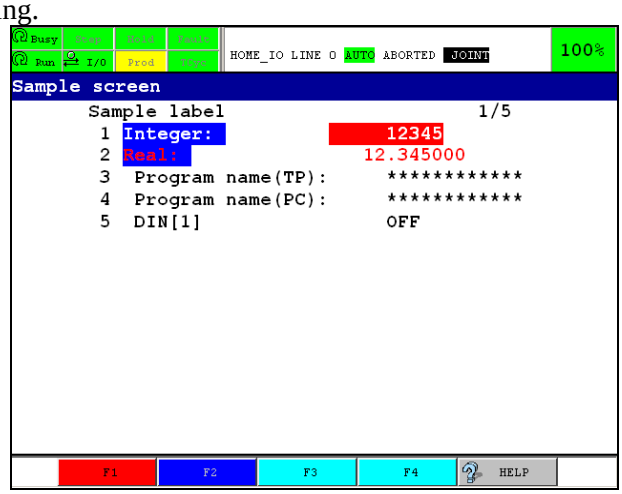
“B” in "-%7B(bool_c)" indicates it is BOOLEAN type. bool_c is any constant. Upper is to display on F5, Lower is F4 ,

$-,bool_c
"FALSE" &new_line
"TRUE"

6.6 DELETE DISPLAY ELEMENTS

Deleting is the inverse of adding way.
1. Delete formtest.pc and formtest.vr from SELECT screen.
2. Change ftstengl.ftx, ftstjapa.ftx and ftstkanj.ftx.
3. Change formtest.kl.
4. Build the dictionaries and formtest.kl, and confirm on ROBOGUIDE.

This section create following.



Change ftstengl.ftx, ftstjapa.ftx and ftstkanj.ftx. Only ftstengl.ftx is changed here as sample.
ftstengl.ftx

```
.kl ftstex
.form
$-,form1
&home &reverse "Sample screen" &standard &new_line
" Sample label " &new_line
*Left side item: background set blue, figure is white.
*Right side: figure is red, background is default.
@3,5&fg_white &bg_blue "Integer: "&bg_dflt &fg_red    @3,23"-%10d(1,32767)"
    &new_line
*Figure stays red, left side background is set blue.
&bg_blue@4,5"Real: "     &bg_dflt       @4,23"-%12f" &fg_dflt &new_line
@5,5" Program name(TP):"     @5,28 "-%12pk(1)" &new_line
```

```
  @6,5" Program name(PC): "    @6,28"-%12pk(2)" &new_line
  @7,5" DIN[1]                 " @7,28"-%7P(io_c)" &new_line
  ^form1_fkey * specifies element which contains
  * function key labels
  ?form1_help * element which contains help
  .endform
  $-,form1_fkey * function key labels
  &bg_red" F1" &new_line    *Background is red
  &bg_blue &fg_white" F2"&fg_dflt    &new_line    *Background is blue, figure is white
  &bg_cyan" F3" &new_line *Back ground is cyan
  " F4" &new_line *No color is specified in this line. Then background stays cyan with
      F3
  &bg_dflt " HELP" *Set background to default
  $-, form1_help * help text
  "Help Line 1" &new_line
  "Help Line 2" &new_line
  "Help Line 3" &new_line
  * You can have a maximum of 48 help lines
  $-, io_c
  " OFF" &new_line
  " ON "
  $-, item_c
  &fg_white &bg_green "item 1"    *item 1, green
  $-
  "item 2"
  $-
  "item 3"
  $-
"¥a"
```

formtest.kl

```
 PROGRAM formtest
 %INCLUDE klevkmsk
 %INCLUDE klevkeys
 %INCLUDE ftstex
 VAR
    l_status:INTEGER
    value_array: ARRAY[5] OF STRING[30]
    change_array:ARRAY[1] OF BOOLEAN --Allows setting size [1] if not use
    inact_array:ARRAY[1] OF BOOLEAN--Allows setting size [1] if not use
    test_int:INTEGER
    test_real:REAL
    prog_name1:STRING[40]
    prog_name2:STRING[40]
    select_item: INTEGER
    def_item :INTEGER
    term_mask:INTEGER
    term_char:INTEGER
    exit_menu:BOOLEAN
    device_stat:INTEGER
```

```
BEGIN
   test_int    = 12345
   test_real = 12.345
   -- There are 7 data to display such as %d in *.ftx
   -- Then value_array is ARRAY[7]
   value_array[1] = 'test_int'    --First data item %10d is test_int of INTEGER
   value_array[2] = 'test_real' --Second %12f is test_real of REAL
   value_array[3] = 'prog_name1'--Third %12pk is prog_name1 of STRING
   value_array[4] = 'prog_name2'--Fourth %12pk is prog_name2 of STRING
   value_array[5] = 'DIN[1]'       --Fifth %7P is BOOLEAN to show DI[1]

   def_item = 1
   term_mask = kc_func_key
   --Force to switch USER2 screen
   FORCE_SPMENU(device_stat, SPI_TPUSER2, 1)
   exit_menu = FALSE
   REPEAT
      --Display and wait for input keys.
      DISCTRL_FORM('ftst', form1, value_array, inact_array, change_array,
                        term_mask, def_item, term_char, l_status)
      IF term_char = KY_NEW_MENU THEN
         --Exit loop if requested.
         exit_menu = TRUE
      else

      ENDIF
   UNTIL exit_menu
END formtest
```

Build dictionary, translate formtest.kl and confirm on ROBOGUIDE. Above screen is displayed.

7 REGISTER SCREEN

7.1 SET FROM SYSTEM VARIABLE SCREEN

It allows KAREL program to register menu. Set the KAREL program name to system variable listed below. It is easy to set it to menu with KAREL Use Support Function. It is option, A05B-2500-J971. Refer chapter 10 KAREL USE SUPPORT FUNCTION.

- $CUSTOMMENU[n].$TITLE
- $CUSTOMMENU[n].$PROG_NAME

n is a value among 1 and 31. The KAREL program is determined to set where by the value of n. However, in case of n is 8, 9, 29, 30 or 31, no menu is set. Following table shows where the KAREL program is set by n.

System Variables	Displayed Menu
$CUSTOMMENU[1]	EDIT F1[INST]
$CUSTOMMENU[2]	EDIT F1[INST]
$CUSTOMMENU[3]	EDIT F2
$CUSTOMMENU[4]	EDIT F3
$CUSTOMMENU[5]	EDIT F4
$CUSTOMMENU[6]	FCTN
$CUSTOMMENU[7]	FCTN
$CUSTOMMENU[10]	UTILITIES
$CUSTOMMENU[11]	UTILITIES
$CUSTOMMENU[12]	TEST CYCLE
$CUSTOMMENU[13]	MANUAL FCTNS
$CUSTOMMENU[14]	MANUAL FCTNS
$CUSTOMMENU[15]	ALARM
$CUSTOMMENU[16]	ALARM
$CUSTOMMENU[17]	I/O
$CUSTOMMENU[18]	I/O
$CUSTOMMENU[19]	SETUP
$CUSTOMMENU[20]	SETUP
$CUSTOMMENU[21]	FILE
$CUSTOMMENU[22]	DATA
$CUSTOMMENU[23]	DATA
$CUSTOMMENU[24]	STATUS
$CUSTOMMENU[25]	STATUS
$CUSTOMMENU[26]	STATUS
$CUSTOMMENU[27]	STATUS
$CUSTOMMENU[28]	SYSTEM

WANING

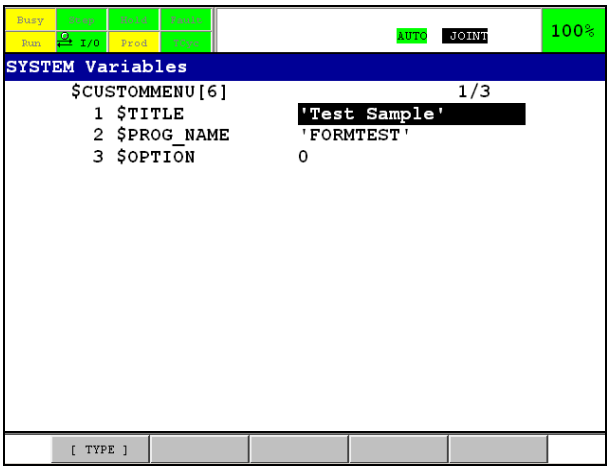
Switch USER2 screen from the KAREL program to display the screen created by KAREL. Displayed screen by KAREL is not allowed to use F1[TYPE]. If you get out from its screen, switch other screen.

Below is to execute formtest.pc from FCTN menu as a sample.

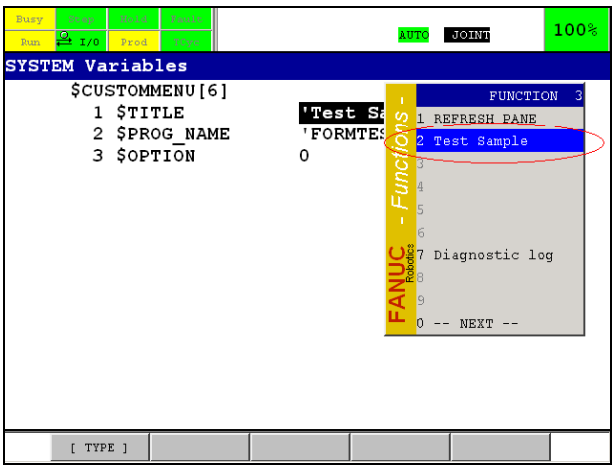
```
PROGRAM formtest
%INCLUDE klevkmsk
%INCLUDE klevkeys
%INCLUDE ftstex
VAR
   l_status:INTEGER
   value_array: ARRAY[5] OF STRING[30]
   change_array:ARRAY[1] OF BOOLEAN --Allows setting size [1] if not use
   inact_array:ARRAY[1] OF BOOLEAN--Allows setting size [1] if not use
   test_int:INTEGER
   test_real:REAL
   prog_name1:STRING[40]
   prog_name2:STRING[40]
   select_item: INTEGER
   def_item :INTEGER
   term_mask:INTEGER
   term_char:INTEGER
   exit_menu:BOOLEAN
   device_stat:INTEGER
BEGIN
   test_int  = 12345
   test_real = 12.345
   -- There are 7 data to display such as %d in *.ftx
   -- Then value_array is ARRAY[7]
   value_array[1] = 'test_int'   --First data item %10d is test_int of INTEGER
   value_array[2] = 'test_real' --Second %12f is test_real of REAL
   value_array[3] = 'prog_name1'--Third %12pk is prog_name1 of STRING
   value_array[4] = 'prog_name2'--Fourth %12pk is prog_name2 of STRING
   value_array[5] = 'DIN[1]'      --Fifth %7P is BOOLEAN to show DI[1]

   def_item = 1
   term_mask = kc_func_key
   --Force to switch USER2 screen
   FORCE_SPMENU(device_stat, SPI_TPUSER2, 1)
   exit_menu = FALSE
   REPEAT
      --Display and wait for input keys.
      DISCTRL_FORM('ftst', form1, value_array, inact_array, change_array,
                   term_mask, def_item, term_char, l_status)
      IF term_char = KY_NEW_MENU THEN
         FORCE_SPMENU(device_stat,SPI_TPHINTS,1)
         --Exit loop if requested.
         exit_menu = TRUE
      else

      ENDIF
   UNTIL exit_menu
END formtest
```

This is the sample to switch HINT screen when the request of switching other screen is received.

Set $CUSTOMMENU[6] as follows in SYSTEM screen.



Select FCTN.



Confirm “Test sample” is added to FCTN menu.

Select “Test sample”, then formtest.pc is executed. Confirm created screen is displayed. Switch the screen such as pushing DATA button. Confirm HINT screen is displayed.

In system variable screen, it dose not allow to display two bytes characters. (It allows inputting.) Confirming input comment, see menu like above.

Next section program for Form Editor allows showing two bytes characters.

7.2 THE SAMPLE TO SET SYSTEM VARIABLES FROM KAREL PROGRAM

Create following dictionary files (cstmengl.ftx, cstmjapa.ftx, cstmkanj.ftx) and KAREL program (cs_set.kl) for Form Editor and Execute.

cstmengl.ftx (English)

```
.kl cstm
.form
$-,form1
&home &reverse "Customizing User menu" &standard &new_line
"$CUSTOMMENU[] Config" &new_line
@3,5 "[1].$TITLE"              @3,23"-%12k" &new_line
@4,5 "[1].$PROG_NAME" @4,23"-%12k"   &new_line
@5,5 "[2].$TITLE"              @5,23"-%12k" &new_line
@6,5 "[2].$PROG_NAME" @6,23"-%12k"   &new_line
@7,5 "[3].$TITLE"              @7,23"-%12k" &new_line
@8,5 "[3].$PROG_NAME" @8,23"-%12k"   &new_line
@9,5 "[4].$TITLE"              @9,23"-%12k" &new_line
@10,5 "[4].$PROG_NAME" @10,23"-%12k"   &new_line
@11,5 "[5].$TITLE"              @11,23"-%12k" &new_line
@12,5 "[5].$PROG_NAME" @12,23"-%12k"   &new_line
@13,5 "[6].$TITLE"              @13,23"-%12k" &new_line
@14,5 "[6].$PROG_NAME" @14,23"-%12k"   &new_line
@15,5 "[7].$TITLE"              @15,23"-%12k" &new_line
@16,5 "[7].$PROG_NAME" @16,23"-%12k"   &new_line
@17,5 "[10].$TITLE"              @17,23"-%12k" &new_line
@18,5 "[10].$PROG_NAME" @18,23"-%12k"   &new_line
@19,5 "[11].$TITLE"              @19,23"-%12k" &new_line
@20,5 "[11].$PROG_NAME" @20,23"-%12k"   &new_line
@21,5 "[12].$TITLE"              @21,23"-%12k" &new_line
@22,5 "[12].$PROG_NAME" @22,23"-%12k"   &new_line
@23,5 "[13].$TITLE"              @23,23"-%12k" &new_line
@24,5 "[13].$PROG_NAME" @24,23"-%12k"   &new_line
@25,5 "[14].$TITLE"              @25,23"-%12k" &new_line
@26,5 "[14].$PROG_NAME" @26,23"-%12k"   &new_line
@27,5 "[15].$TITLE"              @27,23"-%12k" &new_line
@28,5 "[15].$PROG_NAME" @28,23"-%12k"   &new_line
@29,5 "[16].$TITLE"              @29,23"-%12k" &new_line
@30,5 "[16].$PROG_NAME" @30,23"-%12k"   &new_line
@31,5 "[17].$TITLE"              @31,23"-%12k" &new_line
@32,5 "[17].$PROG_NAME" @32,23"-%12k"   &new_line
@33,5 "[18].$TITLE"              @33,23"-%12k" &new_line
@34,5 "[18].$PROG_NAME" @34,23"-%12k"   &new_line
@35,5 "[19].$TITLE"              @35,23"-%12k" &new_line
@36,5 "[19].$PROG_NAME" @36,23"-%12k"   &new_line
@37,5 "[20].$TITLE"              @37,23"-%12k" &new_line
@38,5 "[20].$PROG_NAME" @38,23"-%12k"   &new_line
@39,5 "[21].$TITLE"              @39,23"-%12k" &new_line
@40,5 "[21].$PROG_NAME" @40,23"-%12k"   &new_line
@41,5 "[22].$TITLE"              @41,23"-%12k" &new_line
@42,5 "[22].$PROG_NAME" @42,23"-%12k"   &new_line
@43,5 "[23].$TITLE"              @43,23"-%12k" &new_line
@44,5 "[23].$PROG_NAME" @44,23"-%12k"   &new_line
@45,5 "[24].$TITLE"              @45,23"-%12k" &new_line
@46,5 "[24].$PROG_NAME" @46,23"-%12k"   &new_line
```

```
@47,5 "[25].$TITLE"              @47,23"-%12k" &new_line
@48,5 "[25].$PROG_NAME" @48,23"-%12k"   &new_line
@49,5 "[26].$TITLE"              @49,23"-%12k" &new_line
@50,5 "[26].$PROG_NAME" @50,23"-%12k"   &new_line
@51,5 "[27].$TITLE"              @51,23"-%12k" &new_line
@52,5 "[27].$PROG_NAME" @52,23"-%12k"   &new_line
@53,5 "[28].$TITLE"              @53,23"-%12k" &new_line
@54,5 "[28].$PROG_NAME" @54,23"-%12k"   &new_line
^form1_fkey * specifies element which contains
* function key labels
?form1_help * element which contains help
.endform
$-,form1_fkey * function key labels
"" &new_line
""   &new_line
"" &new_line
"" &new_line
" help"
$-, form1_help * help text
*12345678901234567890123456789012345
* You can have a maximum of 48 help lines
"Title is the name of showing on menu." &new_line
"PROG_NAME is the program name to"&new_line
"be executed." &new_line
"[1], [2} EDIT[INST]" &new_line
"[3] EDIT F2"&new_line
"[4] EDIT F3" &new_line
"[5] EDITF4" &new_line
"[6],[7] FCTN" &new_line
"[10], [11] UTILITIES" &new_line
"[12] TEST CYCLE" &new_line
"[13], [14] MANUAL FCTNS" &new_line
"[15], [16] ALARM" &new_line
"[17], [18] I/O" &new_line
"[19], [20] SETUP" &new_line
"[21] FILE" &new_line
"[22], [23] DATA" &new_line
"[24], [25],[26],[27] STATUS" &new_line
"[28] SYSTEM" &new_line
```

cstmjapa.ftx (for KANA: Legcy TP)

```
.form
$-,form1
&home &reverse "ﾒﾆｭｰ ﾉ ｶｽﾀﾏｲｽﾞ" &standard &new_line
"$CUSTOMMENU[] ｾｯﾃｲ" &new_line
@3,5 "[1].$TITLE"              @3,23"-%12k" &new_line
@4,5 "[1].$PROG_NAME" @4,23"-%12k"   &new_line
@5,5 "[2].$TITLE"              @5,23"-%12k" &new_line
@6,5 "[2].$PROG_NAME" @6,23"-%12k"   &new_line
```

```
@7,5 "[3].$TITLE"              @7,23"-%12k" &new_line
@8,5 "[3].$PROG_NAME" @8,23"-%12k"   &new_line
@9,5 "[4].$TITLE"              @9,23"-%12k" &new_line
@10,5 "[4].$PROG_NAME" @10,23"-%12k"   &new_line
@11,5 "[5].$TITLE"             @11,23"-%12k" &new_line
@12,5 "[5].$PROG_NAME" @12,23"-%12k"   &new_line
@13,5 "[6].$TITLE"             @13,23"-%12k" &new_line
@14,5 "[6].$PROG_NAME" @14,23"-%12k"   &new_line
@15,5 "[7].$TITLE"             @15,23"-%12k" &new_line
@16,5 "[7].$PROG_NAME" @16,23"-%12k"   &new_line
@17,5 "[10].$TITLE"            @17,23"-%12k" &new_line
@18,5 "[10].$PROG_NAME" @18,23"-%12k"   &new_line
@19,5 "[11].$TITLE"            @19,23"-%12k" &new_line
@20,5 "[11].$PROG_NAME" @20,23"-%12k"   &new_line
@21,5 "[12].$TITLE"            @21,23"-%12k" &new_line
@22,5 "[12].$PROG_NAME" @22,23"-%12k"   &new_line
@23,5 "[13].$TITLE"            @23,23"-%12k" &new_line
@24,5 "[13].$PROG_NAME" @24,23"-%12k"   &new_line
@25,5 "[14].$TITLE"            @25,23"-%12k" &new_line
@26,5 "[14].$PROG_NAME" @26,23"-%12k"   &new_line
@27,5 "[15].$TITLE"            @27,23"-%12k" &new_line
@28,5 "[15].$PROG_NAME" @28,23"-%12k"   &new_line
@29,5 "[16].$TITLE"            @29,23"-%12k" &new_line
@30,5 "[16].$PROG_NAME" @30,23"-%12k"   &new_line
@31,5 "[17].$TITLE"            @31,23"-%12k" &new_line
@32,5 "[17].$PROG_NAME" @32,23"-%12k"   &new_line
@33,5 "[18].$TITLE"            @33,23"-%12k" &new_line
@34,5 "[18].$PROG_NAME" @34,23"-%12k"   &new_line
@35,5 "[19].$TITLE"            @35,23"-%12k" &new_line
@36,5 "[19].$PROG_NAME" @36,23"-%12k"   &new_line
@37,5 "[20].$TITLE"            @37,23"-%12k" &new_line
@38,5 "[20].$PROG_NAME" @38,23"-%12k"   &new_line
@39,5 "[21].$TITLE"            @39,23"-%12k" &new_line
@40,5 "[21].$PROG_NAME" @40,23"-%12k"   &new_line
@41,5 "[22].$TITLE"            @41,23"-%12k" &new_line
@42,5 "[22].$PROG_NAME" @42,23"-%12k"   &new_line
@43,5 "[23].$TITLE"            @43,23"-%12k" &new_line
@44,5 "[23].$PROG_NAME" @44,23"-%12k"   &new_line
@45,5 "[24].$TITLE"            @45,23"-%12k" &new_line
@46,5 "[24].$PROG_NAME" @46,23"-%12k"   &new_line
@47,5 "[25].$TITLE"            @47,23"-%12k" &new_line
@48,5 "[25].$PROG_NAME" @48,23"-%12k"   &new_line
@49,5 "[26].$TITLE"            @49,23"-%12k" &new_line
@50,5 "[26].$PROG_NAME" @50,23"-%12k"   &new_line
@51,5 "[27].$TITLE"            @51,23"-%12k" &new_line
@52,5 "[27].$PROG_NAME" @52,23"-%12k"   &new_line
@53,5 "[28].$TITLE"            @53,23"-%12k" &new_line
@54,5 "[28].$PROG_NAME" @54,23"-%12k"   &new_line
^form1_fkey * specifies element which contains
* function key labels
```

```
?form1_help * element which contains help
.endform
$-,form1_fkey * function key labels
"" &new_line
""    &new_line
"" &new_line
"" &new_line
" help"
$-, form1_help * help text
*12345678901234567890123456789012345678901234
* You can have a maximum of 48 help lines
"KAREL ﾌﾟﾛｸﾞﾗﾑ ヲ ﾄﾞﾉ ﾒﾆｭｰ ﾆ ﾄｳﾛｸｽﾙｶﾖ" &new_line
"ｾｯﾃｲ ｼﾏｽ. TITLE ﾆ ﾋｮｳｼﾞ ﾒｲ."&new_line
"PROG_NAME ﾆ ﾌﾟﾛｸﾞﾗﾑﾒｲ ﾖﾆｭｳﾘｮｸ ｼﾃ" &new_line
"ｸﾀﾞｻｲ." &new_line
"[1], [2} ﾍﾝｼｭｳ[ﾒｲﾚｲ]" &new_line
"[3] ﾍﾝｼｭｳ F2"&new_line
"[4] ﾍﾝｼｭｳ F3" &new_line
"[5] ﾍﾝｼｭｳ F4" &new_line
"[6],[7] ﾎｼﾞｮ ﾒﾆｭｰ" &new_line
"[10], [11] ﾕｰﾃｨﾘﾃｨ" &new_line
"[12] ﾃｽﾄ ｼﾞｯｺｳ" &new_line
"[13], [14] ｼｭﾄﾞｳ ｿｳｻ" &new_line
"[15], [16] ｱﾗｰﾑ" &new_line
"[17], [18] I/O" &new_line
"[19], [20] ｾｯﾃｲ" &new_line
"[21] ﾌｧｲﾙ" &new_line
"[22], [23] ﾃﾞｰﾀ" &new_line
"[24], [25],[26],[27] ｼﾞｮｳﾀｲ" &new_line
"[28] ｼｽﾃﾑ" &new_line
```

cstmkanj.ftx (KANJI: for *i*Pendant)

```
.form
$-,form1
&home &reverse 'メニューのカスタマイズ設定' &standard &new_line
'$CUSTOMMENU[] 設定' &new_line
@3,5 "[1].$TITLE"                @3,23"-%12k" &new_line
@4,5 "[1].$PROG_NAME" @4,23"-%12k"    &new_line
@5,5 "[2].$TITLE"                @5,23"-%12k" &new_line
@6,5 "[2].$PROG_NAME" @6,23"-%12k"    &new_line
@7,5 "[3].$TITLE"                @7,23"-%12k" &new_line
@8,5 "[3].$PROG_NAME" @8,23"-%12k"    &new_line
@9,5 "[4].$TITLE"                @9,23"-%12k" &new_line
@10,5 "[4].$PROG_NAME" @10,23"-%12k"    &new_line
@11,5 "[5].$TITLE"                @11,23"-%12k" &new_line
@12,5 "[5].$PROG_NAME" @12,23"-%12k"    &new_line
```

```
@13,5 "[6].$TITLE"              @13,23"-%12k" &new_line
@14,5 "[6].$PROG_NAME" @14,23"-%12k"   &new_line
@15,5 "[7].$TITLE"              @15,23"-%12k" &new_line
@16,5 "[7].$PROG_NAME" @16,23"-%12k"   &new_line
@17,5 "[10].$TITLE"             @17,23"-%12k" &new_line
@18,5 "[10].$PROG_NAME" @18,23"-%12k"   &new_line
@19,5 "[11].$TITLE"             @19,23"-%12k" &new_line
@20,5 "[11].$PROG_NAME" @20,23"-%12k"   &new_line
@21,5 "[12].$TITLE"             @21,23"-%12k" &new_line
@22,5 "[12].$PROG_NAME" @22,23"-%12k"   &new_line
@23,5 "[13].$TITLE"             @23,23"-%12k" &new_line
@24,5 "[13].$PROG_NAME" @24,23"-%12k"   &new_line
@25,5 "[14].$TITLE"             @25,23"-%12k" &new_line
@26,5 "[14].$PROG_NAME" @26,23"-%12k"   &new_line
@27,5 "[15].$TITLE"             @27,23"-%12k" &new_line
@28,5 "[15].$PROG_NAME" @28,23"-%12k"   &new_line
@29,5 "[16].$TITLE"             @29,23"-%12k" &new_line
@30,5 "[16].$PROG_NAME" @30,23"-%12k"   &new_line
@31,5 "[17].$TITLE"             @31,23"-%12k" &new_line
@32,5 "[17].$PROG_NAME" @32,23"-%12k"   &new_line
@33,5 "[18].$TITLE"             @33,23"-%12k" &new_line
@34,5 "[18].$PROG_NAME" @34,23"-%12k"   &new_line
@35,5 "[19].$TITLE"             @35,23"-%12k" &new_line
@36,5 "[19].$PROG_NAME" @36,23"-%12k"   &new_line
@37,5 "[20].$TITLE"             @37,23"-%12k" &new_line
@38,5 "[20].$PROG_NAME" @38,23"-%12k"   &new_line
@39,5 "[21].$TITLE"             @39,23"-%12k" &new_line
@40,5 "[21].$PROG_NAME" @40,23"-%12k"   &new_line
@41,5 "[22].$TITLE"             @41,23"-%12k" &new_line
@42,5 "[22].$PROG_NAME" @42,23"-%12k"   &new_line
@43,5 "[23].$TITLE"             @43,23"-%12k" &new_line
@44,5 "[23].$PROG_NAME" @44,23"-%12k"   &new_line
@45,5 "[24].$TITLE"             @45,23"-%12k" &new_line
@46,5 "[24].$PROG_NAME" @46,23"-%12k"   &new_line
@47,5 "[25].$TITLE"             @47,23"-%12k" &new_line
@48,5 "[25].$PROG_NAME" @48,23"-%12k"   &new_line
@49,5 "[26].$TITLE"             @49,23"-%12k" &new_line
@50,5 "[26].$PROG_NAME" @50,23"-%12k"   &new_line
@51,5 "[27].$TITLE"             @51,23"-%12k" &new_line
@52,5 "[27].$PROG_NAME" @52,23"-%12k"   &new_line
@53,5 "[28].$TITLE"             @53,23"-%12k" &new_line
@54,5 "[28].$PROG_NAME" @54,23"-%12k"   &new_line
^form1_fkey * specifies element which contains
* function key labels
?form1_help * element which contains help
.endform
$-,form1_fkey * function key labels
"" &new_line
""    &new_line
"" &new_line
```

```
"" &new_line
"  ﾍﾙﾌﾟ"
$-, form1_help * help text
*12345678901234567890123456789012345
* You can have a maximum of 48 help lines
'KAREL ﾌﾟﾛｸﾞﾗﾑをどのﾒﾆｭｰに登録するかを' &new_line
'設定します' &new_line
'TITLE に表示名、PROG_NAME にﾌﾟﾛｸﾞﾗﾑ名' &new_line
'を入力して下さい。' &new_line
'[1], [2]　編集画面の F1[命令]ﾒﾆｭｰ' &new_line
'[3]  編集画面の F2 キー' &new_line
'[4]  編集画面の F3 キー' &new_line
'[5]  編集画面の F4 キー' &new_line
'[6],[7]補助ﾒﾆｭｰ' &new_line
'[10], [11]  ﾕｰﾃｨﾘﾃｨ画面' &new_line
'[12]  ﾃｽﾄ実行画面' &new_line
'[13], [14]  手動操作画面' &new_line
'[15], [16]  ｱﾗｰﾑ画面' &new_line
'[17], [18] I/O 画面' &new_line
'[19], [20]  設定画面' &new_line
'[21]  ﾌｧｲﾙ画面' &new_line
'[22], [23]  データ画面' &new_line
'[24], [25],[26],[27]  状態画面' &new_line
'[28]  システム画面' &new_line
```

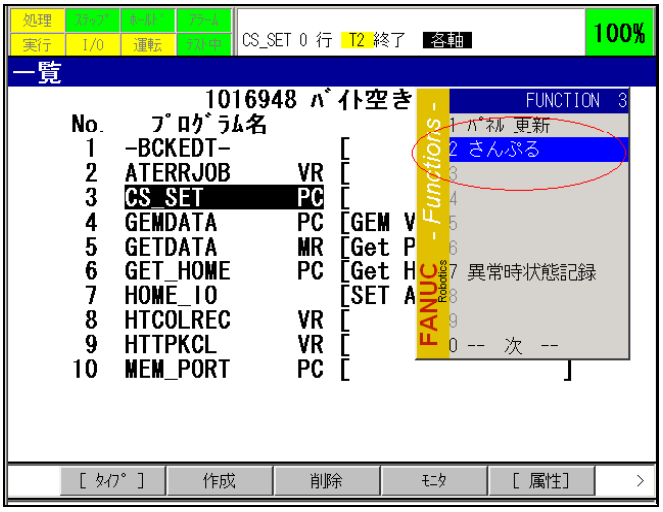
cs_set.kl (KAREL program of setting $CUSTOMMENU[])

```
-- Set $CUSTOMMENU[1]-[7], [10]-[28]
PROGRAM cs_set
%NOLOCKGROUP
%NOPAUSE    = ERROR + COMMAND + TPENABLE
%NOABORT = ERROR + COMMAND + TPENABLE
%NOBUSYLAMP
%INCLUDE klevkmsk
%INCLUDE klevkeys
%INCLUDE cstm
VAR
   l_status:INTEGER
   value_array: ARRAY[52] OF STRING[50]
   change_array:ARRAY[1] OF BOOLEAN
   inact_array:ARRAY[1] OF BOOLEAN
   def_item :INTEGER
   term_mask:INTEGER
   term_char:INTEGER
   exit_menu:BOOLEAN
   device_stat:INTEGER
   i:INTEGER
   n:INTEGER
```

```
    c:INTEGER
    dig: ARRAY[2] OF INTEGER
BEGIN
    i = 1
    n = 0
    c = 0

    FOR i = 1 TO 52 DO
     c =i DIV 2 --division
     n =i MOD 2 --remainder
     IF i < 15 THEN
          c = c + n
          IF (n = 1) THEN
               value_array[i] ='[*system*]$custommenu[' + CHR(48 + c ) +'].$title'
          ELSE
               value_array[i] ='[*system*]$custommenu[' + CHR(48 + c )
     +'].$prog_name'
          ENDIF
     ELSE
          c = c + n + 2
          dig[1] = c MOD 10
          dig[2] = c DIV 10
          IF (n = 1) THEN
               value_array[i] ='[*system*]$custommenu[' + CHR(48 +dig[2])+ CHR(48
     +dig[1]) +'].$title'
          ELSE
               value_array[i] ='[*system*]$custommenu[' +    CHR(48 +dig[2])+ CHR(48
     +dig[1]) +'].$prog_name'
          ENDIF
     ENDIF
    ENDFOR

    def_item = 1
    term_mask = kc_func_key
    --Force to show USER2
    FORCE_SPMENU(device_stat, SPI_TPUSER2, 1)
    exit_menu = FALSE
    REPEAT
       DISCTRL_FORM('cstm', form1, value_array, inact_array, change_array,
                         term_mask, def_item, term_char, l_status)
       IF term_char = KY_NEW_MENU THEN
          --Force to show HINTS
          FORCE_SPMENU(device_stat,SPI_TPHINTS,1)
          exit_menu = TRUE
       else

       ENDIF
    UNTIL exit_menu
END cs_set
```

Display sample after executing:

Allows two bytes characters.



Two bytes characters are in FCTN menu.

8 APPLIED CREATION OF KAREL

8.1 FUNCTION KEY OPERATION

This section explains screen display with function key. For convenience of explanation only English dictionary is described. In case of necessity you should change Japanese (KANA and KANJI) dictionaries.

8.1.1 Pull Up Menu

This subsection explains about pull up menu.
Create following the dictionary file, advcengl.ftx, and the KAREL program, advcsmpl.kl.
advcengl.ftx

```
.kl advceg
.form
$-,form1
&home &reverse "Sample screen" &standard &new_line
" Sample label " &new_line
@3,5" Integer: "      @3,28"-%10d(1,32767)" &new_line
@4,5" Real: "            @4,28"-%12f" &new_line
@5,5" Program name(TP):"     @5,28 "-%12pk(1)" &new_line
@6,5" Program name(PC): "    @6,28"-%12pk(2)" &new_line
@7,5" DIN[1]                 " @7,28"-%7P(io_c)" &new_line
^form1_fkey * specifies element which contains
* function key labels
?form1_help * element which contains help
.endform
$-,form1_fkey * function key labels
"" &new_line
"Sample" &new_line
" F3" &new_line
" F4" &new_line
" HELP"
$-, form1_help * help text
"Help Line 1" &new_line
"Help Line 2" &new_line
"Help Line 3" &new_line
* You can have a maximum of 48 help lines
$-, io_c
" OFF" &new_line
" ON "

$-,f2_menu
"F2-item 1"
$-
"F2-item 2"
$-
"F2-item 3"
```

```
$-
"¥a"

$-, f2_item1
"Selected F2-item 1"
```

advcsmple.kl

```
PROGRAM advcsmpl
%SYSTEM
%NOLOCKGROUP
%NOPAUSE    = ERROR + COMMAND
%NOABORT = ERROR + COMMAND + TPENABLE
%NOBUSYLAMP
%ENVIRONMENT UIF

%INCLUDE klevkmsk -- Key masks
%INCLUDE klevkeys    -- Special keys
%INCLUDE klevccdf    -- Character codes
%INCLUDE advceg

CONST
   DICT_NAME = 'ADVC'

VAR
   l_status:INTEGER
   value_array: ARRAY[5] OF STRING[30]
   change_array:ARRAY[1] OF BOOLEAN --Allows setting size [1] if not use
   inact_array:ARRAY[1] OF BOOLEAN--Allows setting size [1] if not use
   test_int:INTEGER
   test_real:REAL
   prog_name1:STRING[40]
   prog_name2:STRING[40]
   def_item :INTEGER
   term_mask:INTEGER
   term_char:INTEGER
   exit_menu:BOOLEAN
   device_stat:INTEGER
   p_term_char: INTEGER
   p_def_item:INTEGER
   prmpt_win : FILE

BEGIN
   OPEN FILE prmpt_win('RW', 'WD:prmp/tpkb')

   test_int    = 12345
   test_real = 12.345
   -- There are 7 data to display such as %d in *.ftx
   -- Then value_array is ARRAY[7]
   value_array[1] = 'test_int'    --First data item %10d is test_int of INTEGER
```

```
    value_array[2] = 'test_real' --Second %12f is test_real of REAL
    value_array[3] = 'prog_name1'--Third %12pk is prog_name1 of STRING
    value_array[4] = 'prog_name2'--Fourth %12pk is prog_name2 of STRING
    value_array[5] = 'DIN[1]'       --Fifth %7P is BOOLEAN to show DI[1]

    def_item = 1
    term_mask = kc_func_key
    --Force to switch USER2 screen
    FORCE_SPMENU(device_stat, SPI_TPUSER2, 1)
    exit_menu = FALSE
    REPEAT
      --Display and wait for input keys.
      DISCTRL_FORM(DICT_NAME, form1, value_array, inact_array,
     change_array,
                        term_mask, def_item, term_char, l_status)
      SELECT term_char OF
        CASE(KY_NEW_MENU):
          exit_menu = TRUE
        CASE(KY_F2):
          DISCTRL_PLMN(DICT_NAME, f2_menu, 2, p_def_item, p_term_char,
     l_status)
          SELECT p_def_item OF
            CASE(KY_NEW_MENU): exit_menu = TRUE
            CASE(1):
            WRITE prmpt_win(CHR(cc_clear_win))
            WRITE_DICT(prmpt_win, DICT_NAME, f2_item1, l_status)
          ELSE:
          ENDSELECT
      ELSE:
      ENDSELECT
    UNTIL exit_menu
    CLOSE FILE prmpt_win
  END advcsmpl
```

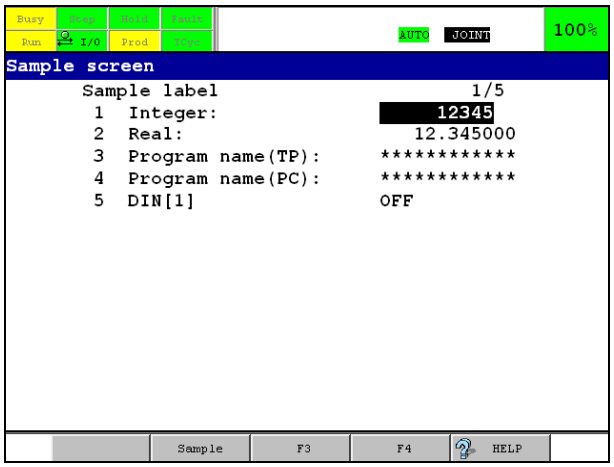
Refer to KAREL Reference Manual (B83144EN-1) about detail of DISCTRL_PLMN built-in function. This function creates pull up menu and controls the movement of the cursor and selecting from menu.

Syntax : DISCTRL_PLMN(dict_name, element_no, ftn_key_no, def_item, term_char, status)
Input/Output Parameters :
[in] dict_name :STRING
[in] element_no :INTEGER
[in] ftn_key_num :INTEGER
[in,out] def_item :INTEGER
[out] term_char :INTEGER
[out] status :INTEGER
%ENVIRONMENT Group : UIF

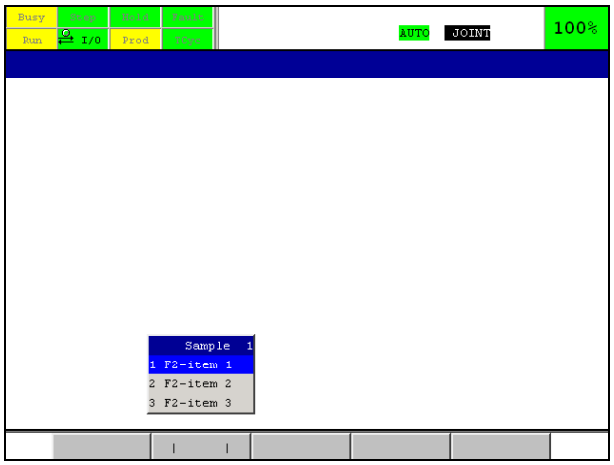
Details:

- The menu data in the dictionary consists of a list of enumerated values that are displayed and selected from a pull-up menu on the teach pendant device. A maximum of 9 values should be used. Each value is a string of up to 12 characters.
- A sequence of consecutive dictionary elements, starting with element_no , define the values. Each value must be put in a separate element, and must not end with &new_line. The characters are assigned the numeric values 1..9 in sequence. The last dictionary element must be "".
- dict_name specifies the name of the dictionary that contains the menu data.
- element_no is the element number of the first menu item within the dictionary.
- ftn_key_num is the function key where the pull-up menu should be displayed.
- def_item is the item that should be highlighted when the menu is entered. 1 specifies the first item. On return, def_item is the item that was currently highlighted when the termination character was pressed.
- term_char receives a code indicating the character that terminated the menu. The codes for key terminating conditions are defined in the include file KLEVKEYS.KL. Keys normally returned are pre-defined constants as follows:
  ky_enter — A menu item was selected
  ky_prev — A menu item was not selected
  ky_new_menu — A new menu was selected
  ky_f1
  ky_f2
  ky_f3
  ky_f4
  ky_f5
- status explains the status of the attempted operation. If not equal to 0, then an error occurred.

Build advcengl.ftc first. Next, build advcsmpl.kl. Execute generated by build advcsmpl.pc.
Following is displayed.



Push F2[Sample] key. Below screen is displayed. All screens are cleared and displayed only F2 key because “noclear” is not specified for Form format. Only “f2_menu” of advcengl.ftx is displayed.



Press PREV key to return sample screen.

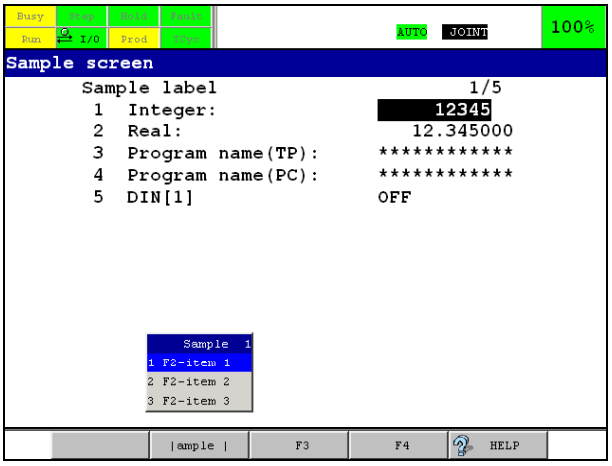
advckanj.ftx

```
.kl advceg
.form noclear
$-,form1
&home &reverse "Sample screen" &standard &new_line
" Sample label " &new_line
@3,5" Integer: "      @3,28"-%10d(1,32767)" &new_line
@4,5" Real: "            @4,28"-%12f" &new_line
@5,5" Program name(TP):"      @5,28 "-%12pk(1)" &new_line
@6,5" Program name(PC): "    @6,28"-%12pk(2)" &new_line
@7,5" DIN[1]                 " @7,28"-%7P(io_c)" &new_line
^form1_fkey * specifies element which contains
* function key labels
?form1_help * element which contains help
.endform
$-,form1_fkey * function key labels
"" &new_line
"Sample" &new_line
" F3" &new_line
" F4" &new_line
" HELP"
$-, form1_help * help text
"Help Line 1" &new_line
"Help Line 2" &new_line
"Help Line 3" &new_line
* You can have a maximum of 48 help lines
$-, io_c
" OFF" &new_line
" ON "

$-,f2_menu
"F2-item 1"
$-
"F2-item 2"
$-
"F2-item 3"
```

```
$-
"¥a"

$-, f2_item1
"Selected F2-item 1"
```

“noclear” is specified like above. Then all screens are not cleared automatically in case of displaying menu or other screens. Therefore you must clear screens manually. If you do not clear screens manually, displayed contents may be remained when you switch to other screens. It is not garbled characters. Be carefully when you specify “noclear”.

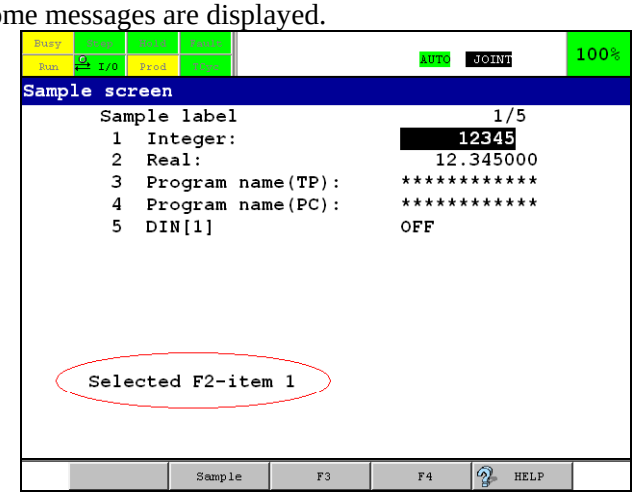
A maximum of 9 values should be used. Each value is a string of up to 12 characters. Refer KAREL Reference Manual (B-83144EN-1) Appendix A (DISCTRL_PLMN part).



Screens are not cleared and displayed function key items when “noclear” is specified like above.

8.1.2 Treatment of Selecting Pull Up Menu

This subsection explains to create process after selecting F2[Sample] – “F2-Item 1”. For example, after selecting function key, some messages are displayed.



Change the dictionary file and the KAREL program to display messages.

advcengl.ftx

```
.kl advceg
.form noclear
$-,form1
&home &reverse "Sample screen" &standard &new_line
" Sample label " &new_line
@3,5" Integer: "      @3,28"-%10d(1,32767)" &new_line
@4,5" Real: "            @4,28"-%12f" &new_line
@5,5" Program name(TP):"      @5,28 "-%12pk(1)" &new_line
@6,5" Program name(PC): "    @6,28"-%12pk(2)" &new_line
@7,5" DIN[1]                " @7,28"-%7P(io_c)" &new_line
^form1_fkey * specifies element which contains
* function key labels
?form1_help * element which contains help
.endform
$-,form1_fkey * function key labels
"" &new_line
"Sample" &new_line
" F3" &new_line
" F4" &new_line
" HELP"
$-, form1_help * help text
"Help Line 1" &new_line
"Help Line 2" &new_line
"Help Line 3" &new_line
* You can have a maximum of 48 help lines
$-, io_c
" OFF" &new_line
" ON "

$-,f2_menu
"F2-item 1"
$-
"F2-item 2"
$-
"F2-item 3"
$-
"¥a"

$-, f2_item1
"Selected F2-item 1"
```

advcsmple.kl

```
PROGRAM advcsmpl
%SYSTEM
%NOLOCKGROUP
%NOPAUSE    = ERROR + COMMAND
%NOABORT = ERROR + COMMAND + TPENABLE
%NOBUSYLAMP
```

```
%ENVIRONMENT UIF

%INCLUDE klevkmsk -- Key masks
%INCLUDE klevkeys    -- Special keys
%INCLUDE klevccdf    -- Character codes
%INCLUDE advceg

CONST
   DICT_NAME = 'ADVC'

VAR
   l_status:INTEGER
   value_array: ARRAY[5] OF STRING[30]
   change_array:ARRAY[1] OF BOOLEAN --Allows setting size [1] if not use
   inact_array:ARRAY[1] OF BOOLEAN--Allows setting size [1] if not use
   test_int:INTEGER
   test_real:REAL
   prog_name1:STRING[40]
   prog_name2:STRING[40]
   def_item :INTEGER
   term_mask:INTEGER
   term_char:INTEGER
   exit_menu:BOOLEAN
   device_stat:INTEGER
   p_term_char: INTEGER
   p_def_item:INTEGER
   prmpt_win : FILE

BEGIN
   OPEN FILE prmpt_win('RW', 'WD:prmp/tpkb')

   test_int    = 12345
   test_real = 12.345
   -- There are 7 data to display such as %d in *.ftx
   -- Then value_array is ARRAY[7]
   value_array[1] = 'test_int'    --First data item %10d is test_int of INTEGER
   value_array[2] = 'test_real' --Second %12f is test_real of REAL
   value_array[3] = 'prog_name1'--Third %12pk is prog_name1 of STRING
   value_array[4] = 'prog_name2'--Fourth %12pk is prog_name2 of STRING
   value_array[5] = 'DIN[1]'       --Fifth %7P is BOOLEAN to show DI[1]

   def_item = 1
   term_mask = kc_func_key
   --Force to switch USER2 screen
   FORCE_SPMENU(device_stat, SPI_TPUSER2, 1)
   exit_menu = FALSE
   REPEAT
      --Display and wait for input keys.
      DISCTRL_FORM(DICT_NAME, form1, value_array, inact_array,
    change_array,
```

```
                    term_mask, def_item, term_char, l_status)
    SELECT term_char OF
      CASE(KY_NEW_MENU):
        exit_menu = TRUE
      CASE(KY_F2):
        DISCTRL_PLMN(DICT_NAME, f2_menu, 2, p_def_item, p_term_char,
  l_status)
        SELECT p_def_item OF
          CASE(KY_NEW_MENU): exit_menu = TRUE
          CASE(1):
          WRITE prmpt_win(CHR(cc_clear_win))
          WRITE_DICT(prmpt_win, DICT_NAME, f2_item1, l_status)
        ELSE:
        ENDSELECT
    ELSE:
    ENDSELECT
  UNTIL exit_menu
  CLOSE FILE prmpt_win
END advcsmpl
```

In this sample FILE type variable, prmpt_win, is defined and cleared prompt line to display messages to prompt line. Then defined dictionary file output messages.

Refer to KAREL Reference Manu Appendix A (WRITE_DICT part).
Syntax : WRITE_DICT(file_var, dict_name, element_no, status)
Input/Output Parameters:

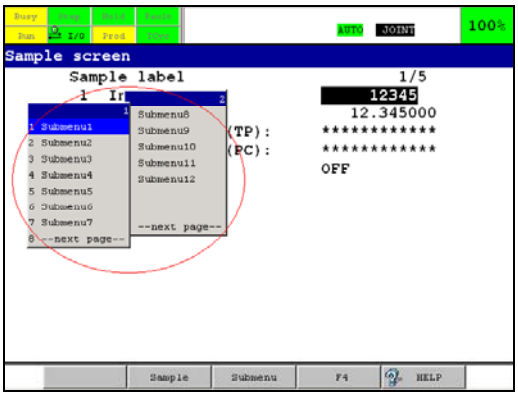
[in] file_var :FILE
[in] dict_name :STRING
[in] element_no :INTEGER
[out] status :INTEGER
%ENVIRONMENT Group :PBCORE
Details:

- file_var must be opened to the window where the dictionary text is to appear.
- dict_name specifies the name of the dictionary from which to write.
- element_no specifies the element number to write. This number is designated with a “$” in the dictionary file.
- status explains the status of the attempted operation. If not equal to 0, then an error occurred writing the element from the dictionary file.

It allows WRITE_DICT to debug effectively when creating KAREL programs and to pay attention to the operator.

8.1.3 Sub Menu

It is allowed to create sub menu with the same way as with pull up menu. Following is the sample of selecting F3[Submenu].



A maximum of pull up menu is 9 but sub menu is 35. Above sample shows 12 elements. To create sub menu you should use DISCTRL_SBMN. Refer to KAREL Reference Manual to know detail of it.

Syntax : DISCTRL_SBMN(dict_name, element_no, def_item, term_char, status)

Input/Output Parameters :
[in] dict_name :STRING
[in] element_no :INTEGER
[in,out] def_item :INTEGER
[out] term_char :INTEGER
[out] status :INTEGER
%ENVIRONMENT Group : UIF

Details:

- The menu data in the dictionary consists of a list of enumerated values that are displayed and selected from the 'subm' subwindow on the Teach Pendant device. There can be up to 5 subwindow pages, for a maximum of 35 values. Each value is a string of up to 16 characters. If 4 or less enumerated values are used, then each string can be up to 40 characters.
- A sequence of consecutive dictionary elements, starting with element_no , define the values. Each value must be put in a separate element, and must not end with &new_line. The characters are assigned the numeric values 1..35 in sequence. The last dictionary element must be "".
- dict_name specifies the name of the dictionary that contains the menu data.
- element_no is the element number of the first menu item within the dictionary.
- def_item is the item that should be highlighted when the menu is entered. 1 specifies the first item. On return, def_item is the item that was currently highlighted when the termination character was pressed.
- term_char receives a code indicating the character that terminated the menu. The codes for key terminating conditions are defined in the include file KLEVKEYS.KL. Keys normally returned are pre-defined constants as follows:
  ky_enter — A menu item was selected
  ky_prev — A menu item was not selected
  ky_new_menu — A new menu was selected
  ky_f1
  ky_f2
  ky_f3
  ky_f4
  ky_f5
- status explains the status of the attempted operation. If not equal to 0, then an error occurred.

Changing the dictionary file and the KAREL program allows creating displaying sample.

advcengl.ftx

```
.kl advceg
.form noclear
$-,form1
&home &reverse "Sample screen" &standard &new_line
" Sample label " &new_line
@3,5" Integer: "     @3,28"-%10d(1,32767)" &new_line
@4,5" Real: "           @4,28"-%12f" &new_line
@5,5" Program name(TP):"     @5,28 "-%12pk(1)" &new_line
@6,5" Program name(PC): "    @6,28"-%12pk(2)" &new_line
@7,5" DIN[1]               " @7,28"-%7P(io_c)" &new_line
^form1_fkey * specifies element which contains
* function key labels
?form1_help * element which contains help
.endform
$-,form1_fkey * function key labels
"" &new_line
"Sample" &new_line
"Submenu" &new_line
" F4" &new_line
" HELP"
$-, form1_help * help text
"Help Line 1" &new_line
"Help Line 2" &new_line
"Help Line 3" &new_line
* You can have a maximum of 48 help lines
$-, io_c
" OFF" &new_line
" ON "

$-,f2_menu
"F2-item 1"
$-
"F2-item 2"
$-
"F2-item 3"
$-
"¥a"

$-, f2_item1
"Selected F2-item 1"

$-, f3_submn
"Submenu1"
$-
"Submenu2"
$-
"Submenu3"
$-
"Submenu4"
$-
```

```
"Submenu5"
$-
"Submenu6"
$-
"Submenu7"
$-
"Submenu8"
$-
"Submenu9"
$-
"Submenu10"
$-
"Submenu11"
$-
"Submenu12"
$-
"¥a"
```

advcsmpl.kl

```
PROGRAM advcsmpl
%SYSTEM
%NOLOCKGROUP
%NOPAUSE   = ERROR + COMMAND
%NOABORT = ERROR + COMMAND + TPENABLE
%NOBUSYLAMP
%ENVIRONMENT UIF

%INCLUDE klevkmsk -- Key masks
%INCLUDE klevkeys   -- Special keys
%INCLUDE klevccdf   -- Character codes
%INCLUDE advceg

CONST
  DICT_NAME = 'ADVC'

VAR
  l_status:INTEGER
  value_array: ARRAY[5] OF STRING[30]
  change_array:ARRAY[1] OF BOOLEAN --Allows setting size [1] if not use
  inact_array:ARRAY[1] OF BOOLEAN--Allows setting size [1] if not use
  test_int:INTEGER
  test_real:REAL
  prog_name1:STRING[40]
  prog_name2:STRING[40]
  def_item :INTEGER
  term_mask:INTEGER
  term_char:INTEGER
  exit_menu:BOOLEAN
  device_stat:INTEGER
  p_term_char: INTEGER
  p_def_item:INTEGER
  prmpt_win : FILE

BEGIN
  OPEN FILE prmpt_win('RW', 'WD:prmp/tpkb')

  test_int   = 12345
  test_real = 12.345
  -- There are 7 data to display such as %d in *.ftx
  -- Then value_array is ARRAY[7]
  value_array[1] = 'test_int'   --First data item %10d is test_int of INTEGER
  value_array[2] = 'test_real' --Second %12f is test_real of REAL
  value_array[3] = 'prog_name1'--Third %12pk is prog_name1 of STRING
  value_array[4] = 'prog_name2'--Fourth %12pk is prog_name2 of STRING
  value_array[5] = 'DIN[1]'       --Fifth %7P is BOOLEAN to show DI[1]

  def_item = 1
  term_mask = kc_func_key
```

```
    --Force to switch USER2 screen
    FORCE_SPMENU(device_stat, SPI_TPUSER2, 1)
    exit_menu = FALSE
    REPEAT
      --Display and wait for input keys.
      DISCTRL_FORM(DICT_NAME, form1, value_array, inact_array,
     change_array,
                        term_mask, def_item, term_char, l_status)
      SELECT term_char OF
        CASE(KY_NEW_MENU):
          exit_menu = TRUE
        CASE(KY_F2):
          DISCTRL_PLMN(DICT_NAME, f2_menu, 2, p_def_item, p_term_char,
     l_status)
          SELECT p_def_item OF
            CASE(KY_NEW_MENU): exit_menu = TRUE
            CASE(1):
            WRITE prmpt_win(CHR(cc_clear_win))
            WRITE_DICT(prmpt_win, DICT_NAME, f2_item1, l_status)
          ELSE:
          ENDSELECT
        CASE(KY_F3):
          p_def_item = 1
          DISCTRL_SBMN(DICT_NAME, f3_submn, p_def_item, p_term_char,
     l_status)
      ELSE:
      ENDSELECT
    UNTIL exit_menu
    CLOSE FILE prmpt_win
END advcsmpl
```

9 DEBUG KAREL PROGRAMS

Following is the ways of debug KAREL programs.

- Use WRITE statement
- Confirm KAREL variable
- Single step

Insert WRITE statement into the position to confirm KAREL treatment. Specified data is displayed on USER screen of teach pendant as executing WRITE statement. Then it allows confirming the KAREL program treatment.

NOTE
WRITE statement gets processing speed of KAREL program lower. You must remove WRITE statement before product line is run after debugging completely.

KAREL global variables are shown in DATA screen. Select the KAREL program to see its global variables and switch DATA screen. Push F1[TYPE] and select “KAREL Vars” or “KAREL Posns” to see them.

9.1 DEBUG WITH WRITE STATEMENT

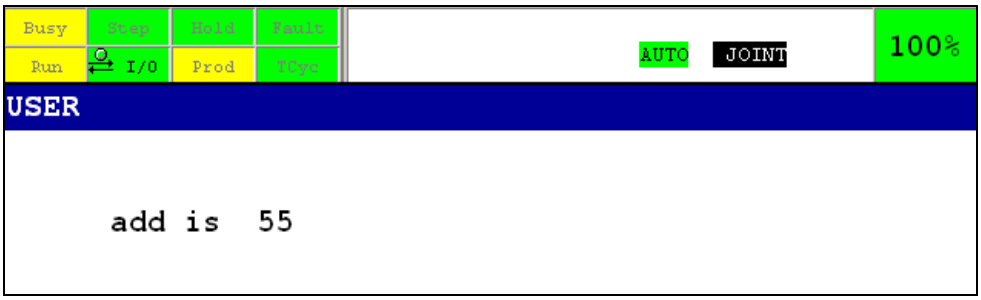
9.1.1 Sample to See the Value of Variables on USER Screen

Following is the sample to see variable, add, on USER screen.
WRITE TPDISPLAY(CHR(cc_clear_win),CHR(cc_home))
WRITE TPDISPLAY('add is ',add,CR)

```
--------------------------------------------------------------------------------
PROGRAM sample
--------------------------------------------------------------------------------
%NOBUSYLAMP
%NOLOCKGROUP
%NOPAUSE = ERROR + COMMAND + TPENABLE
%NOABORT = ERROR + COMMAND
%INCLUDE klevccdf
CONST
   START_NUM = 1
   FINISH_NUM   = 10
VAR
   STATUS : INTEGER
   result : INTEGER
--------------------------------------------------------------------------------
ROUTINE sum( from_num: INTEGER; to_num:INTEGER) : INTEGER
--------------------------------------------------------------------------------
VAR
   idx : INTEGER
   add : INTEGER
BEGIN
   idx = 0
```

```
   add = 0
   FOR idx = from_num TO to_num DO
      add = add + idx
   ENDFOR
   WRITE TPDISPLAY(CHR(cc_clear_win),CHR(cc_home)) -- Clear USER screen
   WRITE TPDISPLAY('add is ',add,CR) --Show add value to USER
   RETURN(add)
END sum
-----------------------------------------------------------------------------
BEGIN
-----------------------------------------------------------------------------
   result = sum( START_NUM, FINISH_NUM)
   SET_INT_REG(1,result,STATUS)

END sample
```

After executing above sample and switching USER screen (Select Menu -> USER) following message is displayed.



Describing TPDISPLAY after WRITE statement means to output the data to USER. Include klevccdf and execute
“WRITE TPDISPLAY(CHR(cc_clear_win),CHR(cc_home))”. This is to clear USER screen.
CR of “WRITE TPDISPLAY('add is ',add,CR)” means line feed.

9.1.2 Sample to Output to Specified File

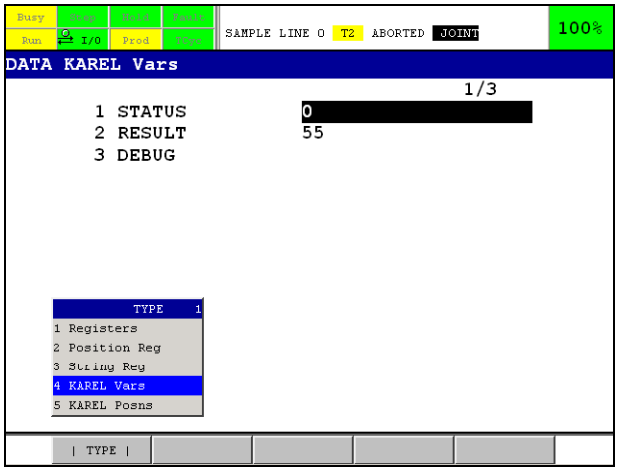
```
-----------------------------------------------------------------------------
PROGRAM sample
-----------------------------------------------------------------------------
%NOBUSYLAMP
%NOLOCKGROUP
%NOPAUSE = ERROR + COMMAND + TPENABLE
%NOABORT = ERROR + COMMAND
%INCLUDE klevccdf
CONST
   START_NUM = 1
   FINISH_NUM    = 10
VAR
   STATUS : INTEGER
   result : INTEGER
   debug:FILE --Add file variable
-----------------------------------------------------------------------------
ROUTINE sum( from_num: INTEGER; to_num:INTEGER) : INTEGER
```

```
--------------------------------------------------------------------------------
VAR
   idx : INTEGER
   add : INTEGER
BEGIN
   idx = 0
   add = 0
   FOR idx = from_num TO to_num DO
      add = add + idx
   ENDFOR
   WRITE debug('add is ', add, CR) -- Output add variable to MC:¥debug.txt
   RETURN(add)
END sum
--------------------------------------------------------------------------------
BEGIN
--------------------------------------------------------------------------------
   OPEN FILE debug('RW','MC:¥debug.txt') --Open MC:¥debug.txt with RW mode

   result = sum( START_NUM, FINISH_NUM)
   SET_INT_REG(1,result,STATUS)

   CLOSE FILE debug --Close file

END sample
```

It defines FILE variables (in this sample “debug”). It opens file with “Read Write” mode and outputs the data to the file with WRITE statement. Finally it closes file. “add is 55” is output to MC:¥debug.txt.

9.2 CONFIRM KAREL VARIABLES

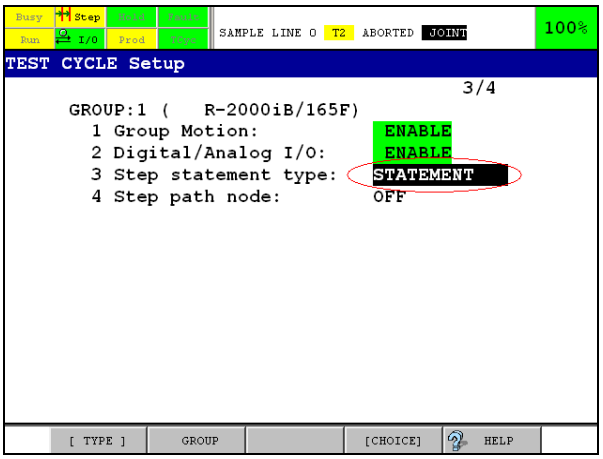
Execute the KAREL program, sample.kl, used previous subsection. Switch DATA screen (Push DATA key on teach pendant or select Menu -> NEXT -> DATA). Push F1[TYPE] and select “KAREL Vars” or “KAREL Posns” to see them. KAREL variables are displayed as follows.



Only global variables are shown on DATA screen. Local variables, defined in local routine, are never shown.

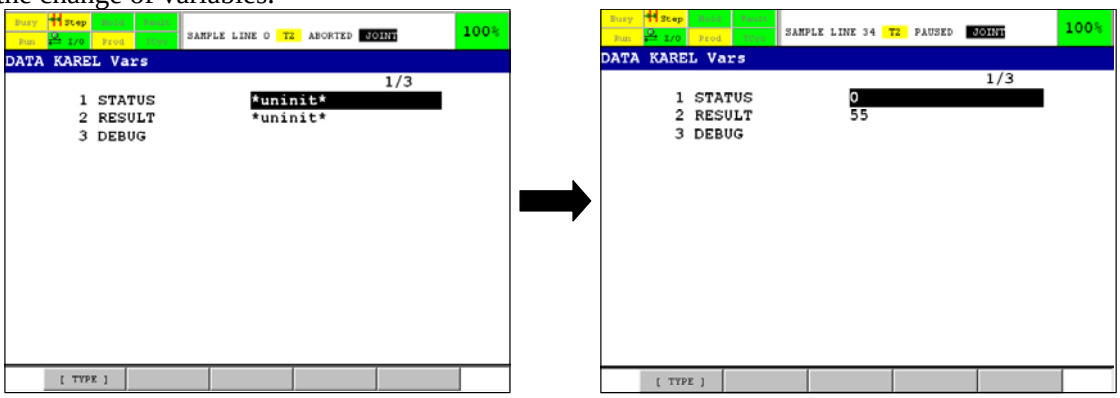
9.3 CONFIRM WITH SINGLE STEP

Single step is enabled in KAREL program as long as not specified %SYSTEM attribute. It allows confirming KAREL variables every step of KAREL statement and display of WRITE statement. Then they help debugging. How single step is enabled is to set “Step statement type” to STATEMENT in TEST CYCLE Setup screen (Menu -> TEST CYCLE). Below is the display sample.



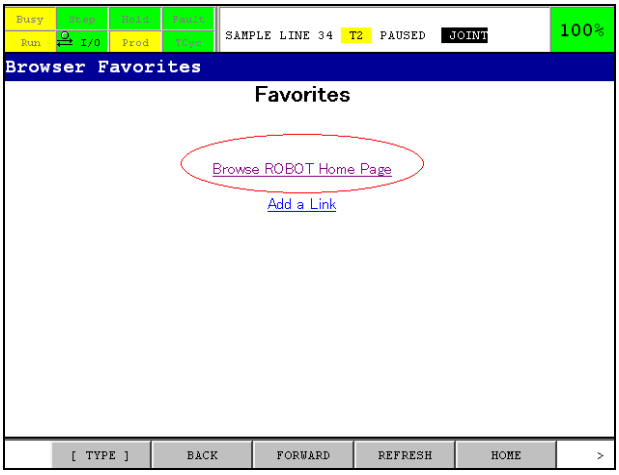
Select the KAREL file (ex. sample.pc) from SELECT screen and execute it with single step. The KAREL program is paused at every KAREL instruction. It allows executing status to see in DATA screen at every step.

Ex. See the change of variables.

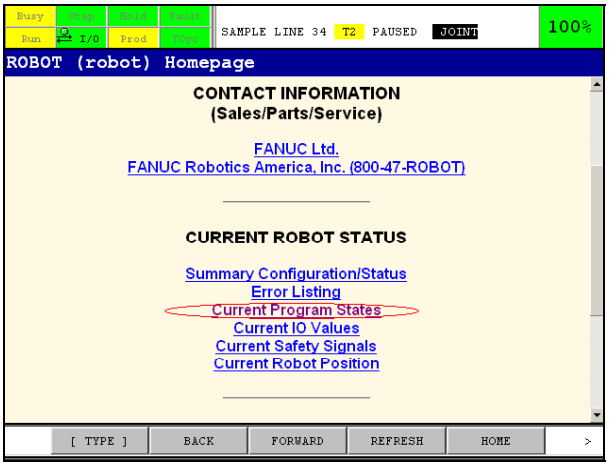


9.4 CONFIRM EXECUTION STATUS OF KAREL PROGRAM

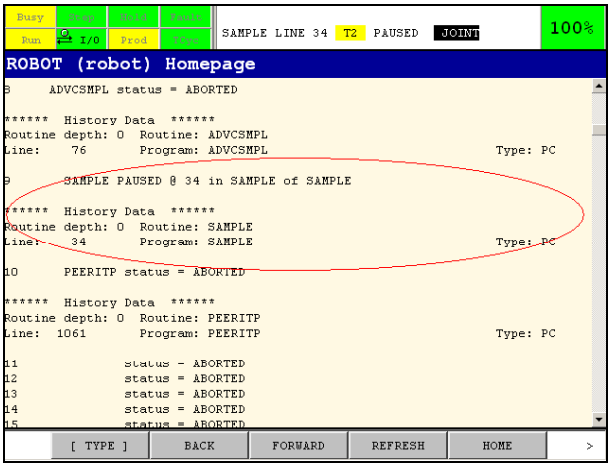
It allows confirming execution status of KAREL program with *i*Pendant. Both of actual and virtual robots are allowed. Select Menu -> NEXT -> BROWSER. Select ROBOT in center of the screen.



Select “Current Program States”.



It shows program status.



Scrolling below allows confirming.
Push Shift key and down arrow key to scroll a page.

10 KAREL USE SUPPORT FUNCTION

This chapter explains about KAREL Use Support Function, option J971.

10.1 OVERVIEW

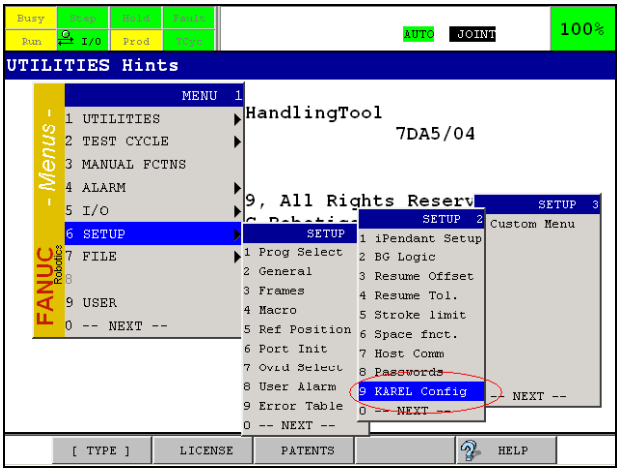
This function supplies the interface for KAREL execution, abort, status output and so on. It enables following operations.

- Executing or aborting KAREL programs.
- Auto running the KAREL programs at the time of power on.
- Periodically monitor the specified KAREL program and outputting the status.
- Set the configuration of custom menu setting.

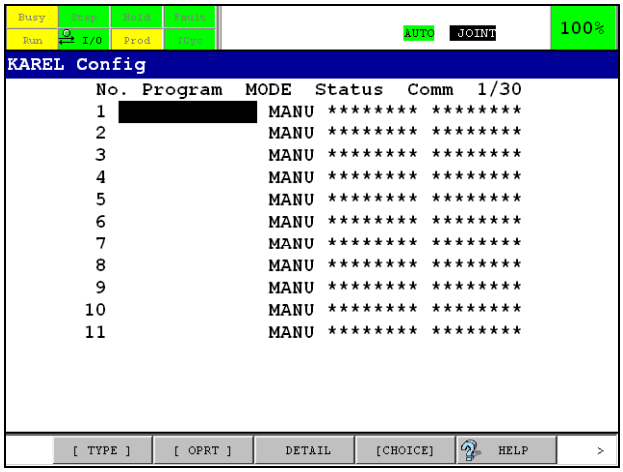
10.2 KAREL CONFIG

10.2.1 Start the KAREL Config Screen

Select menu – setup – KAREL Config.

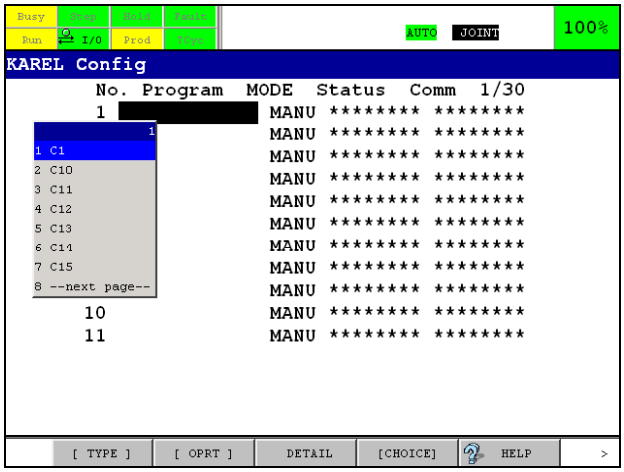


Like following screen is displayed.

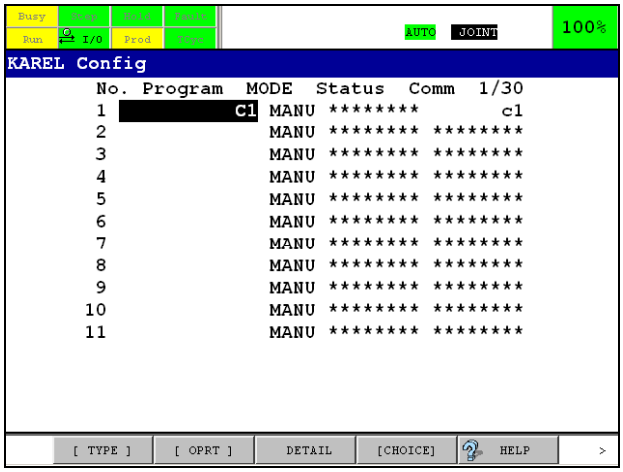


10.2.2 Use KAREL Config Screen

Select F4[CHOICE], then you can see the list of KAREL program to be registered to run.



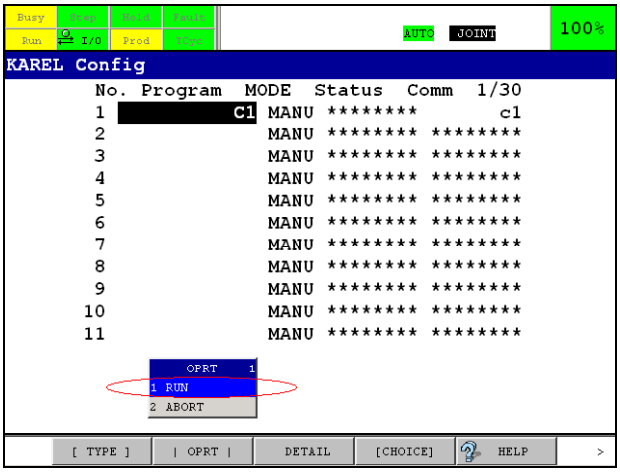
In this documentation, C1 is used as the sample to explain. Set the cursor onto the C1, and press ENTER key.



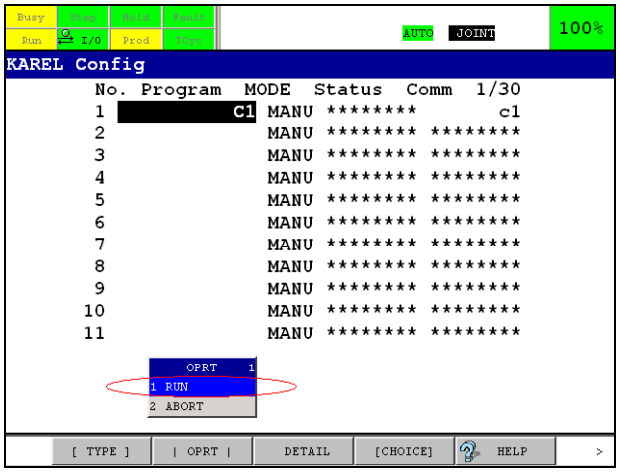
There are four status, running, paused, aborted and ********. ******** means that status is not other 3 ones.
Comm is the comment that is described in the KAREL source file as %COMMENT.

10.2.3 Run KAREL Programs

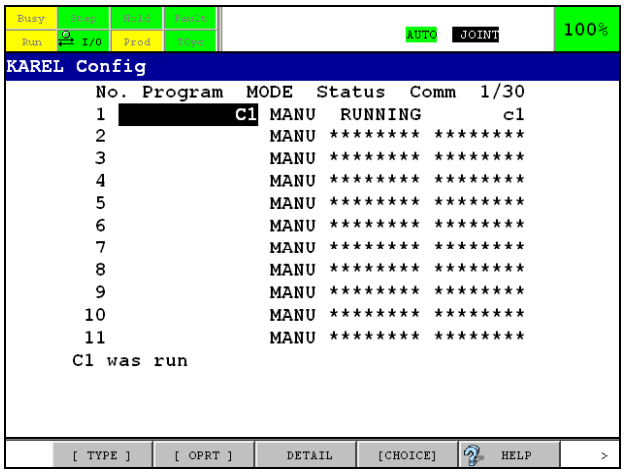
Set the cursor onto C1 in Program row and press F2[OPRT].



Select 1. RUN. Then confirmation message is displayed.

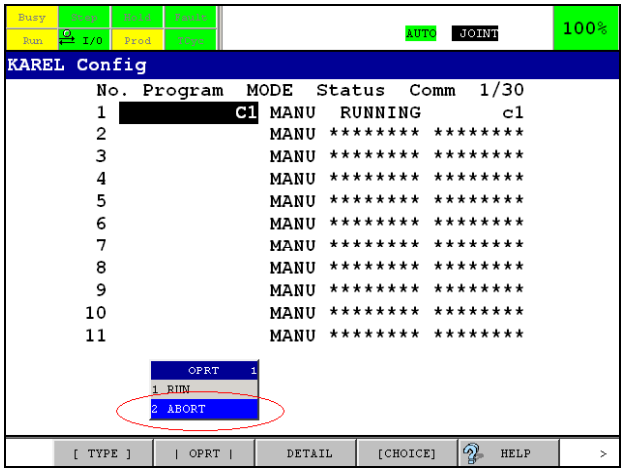


Program is run after selecting F4[YES].

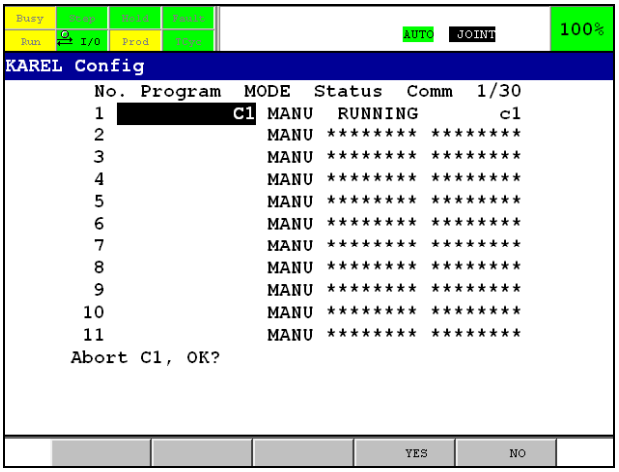


10.2.4 Abort KAREL Programs

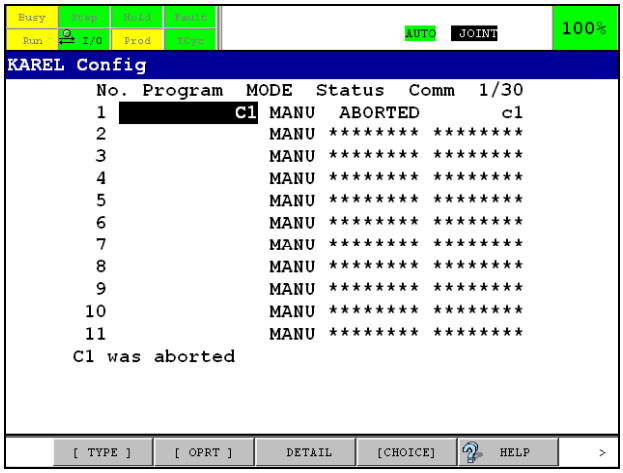
Set the cursor onto C1 in Program row and press F2[OPRT].



Set the cursor onto 2. ABORT and press ENTER key.
Confirmation message is displayed.

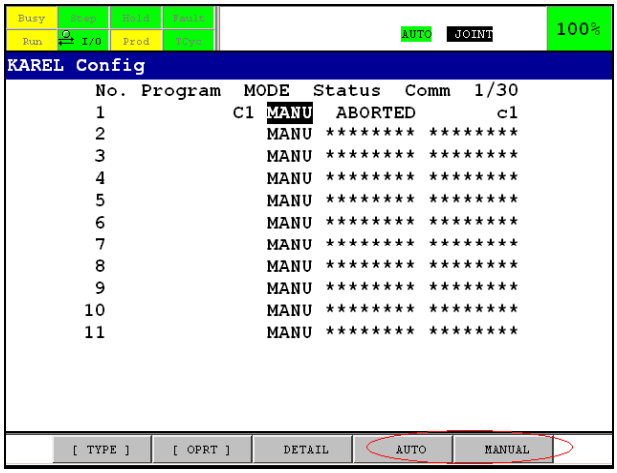


Select F4[YES], then KAREL program is aborted.

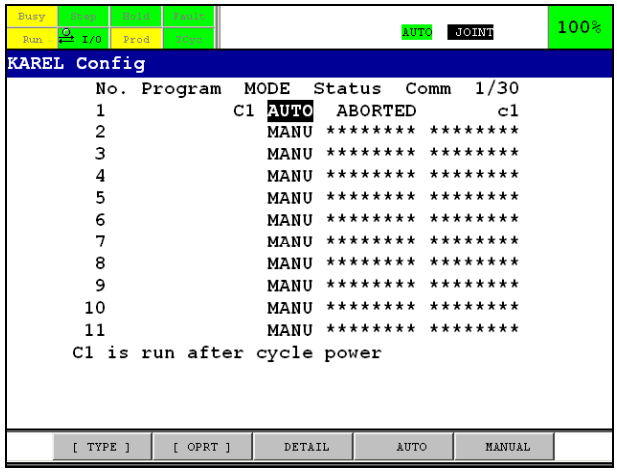


10.2.5 Start Mode Config

Set the cursor onto MODE row. F4 and F5 keys are turns to AUTO and MANUAL



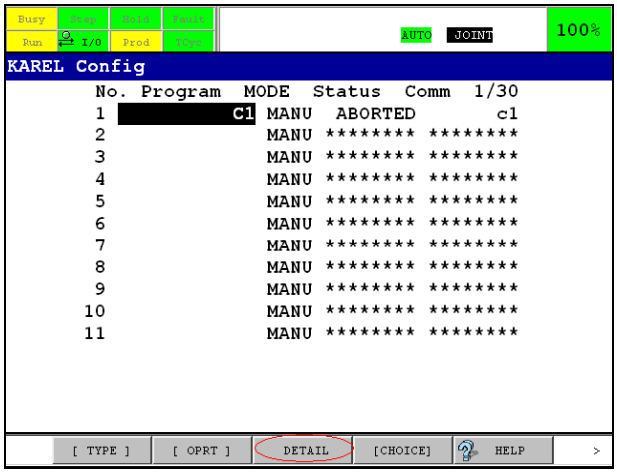
Default is MANUAL. If you select AUTO, KAREL program is run after power on the controller



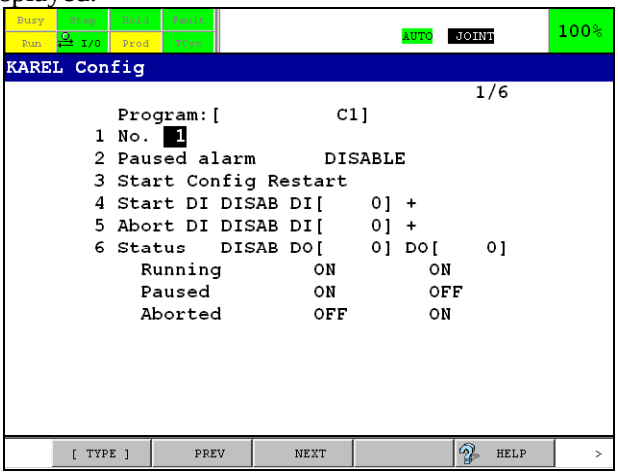
There is a limit that the number of running the KAREL simultaneously. It depends on the configuration of the robot system. “PROG-014 Max task number exceed” will be posted when the number of running KAREL is going to be over max number.

10.2.6 Detail of KAREL Config

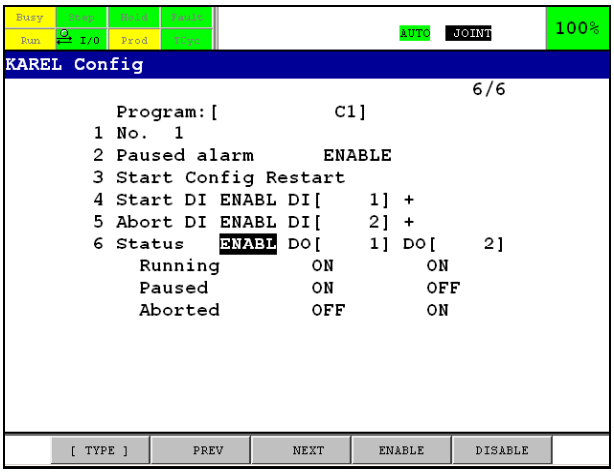
Press F3[DETAIL].



Like following screen is displayed.



ITEM	COMMENTS	DEFAULT
Paused alarm	If this is enable, warning is posted when specified program, ex C1, is paused or aborted. In the case of paused, “SYST 294 Paused (program name)” and “SYST 296 Specified KAREL was paused” are posted. In the case of aborted, “SYST 295 Aborted (program name)” and “SYST 297 Specified KAREL was paused” are posted.	DISABLE
Start Config	Sets the mode of restart. “Restart” is once aborting the specified program and start from the top of the program. “Resume” is not once aborted. Specified program is restarted from the paused point.	Restart
Start DI	Sets the start operation. Detecting the rising edge (+) or falling edge (-) of specified DI starts the specified program. When “Start DI” is set enable, DI port number and Up or Down can not be changed. If you would like to set them, please set “Start DI” disable once. After setting them, please set it enable. Then this configuration is enabled after cycle power.	DISABLE
Abort DI	Sets the abort operation. How to set is the same of “ Start DI”.	DISABLE
Status	To monitor the specified program and output its status with two DOs. As the default, program is monitored every 100msec. Its config is enabled after cycle power. Following is the sample of configuration.	DISABLE



On this sample, DO[1] and DO[2] are ON while program C1 is running. DO[1] is ON and DO[2] is OFF while it is pausing.
DO[1] is OFF and DO[2] is ON while aborting. If status is not each three status, DO[1] and DO[2] are both OFF.

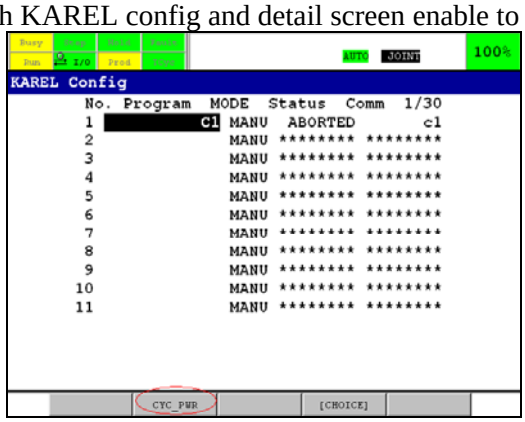
To switch previous screen, press F2[PREV]. In this case, screen is switched to No.30.
To switch next detail screen, press F3[NEXT]. In this case, screen is switched to No.2.

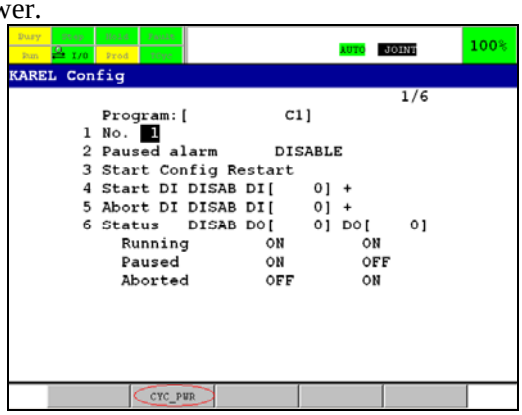
10.2.6.1 Limitation and caution of KAREL config

- Status is monitored at a constant frequency. Thus status is not changed at a real time. For example, program status is changed from running to paused, and aborted in one monitoring cycle, paused status is not detected at next monitoring cycle. Therefore status is changed from running to aborted in this case.
- Monitoring constant frequency can be changed. You can change it with system variable $KRL_NUM.$UPDT_TIME. It is able to change in the range 8 to 5000 msec. If you set it out of its range under 8 or above 5000, it is automatically set to 8 or 5000msec.
- In the case of above sample, after you set the detail of C1, return to KAREL Config screen with PREV key and change the registered No.1 program from C1 to other program, such as C2, C1 does not start if DI[1] is turned ON. C2 is run instead. For prevention of miss operation, please do not change program after set the detail.

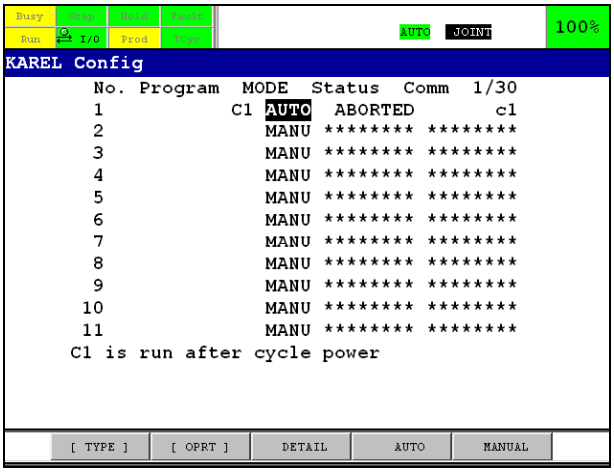
10.2.7 Cycle Power

Both KAREL config and detail screen enable to cycle power.





F2[CYC_PWR] is displayed when you press NEXT key. Set the TP to enable and press F2[CYC_PWR].

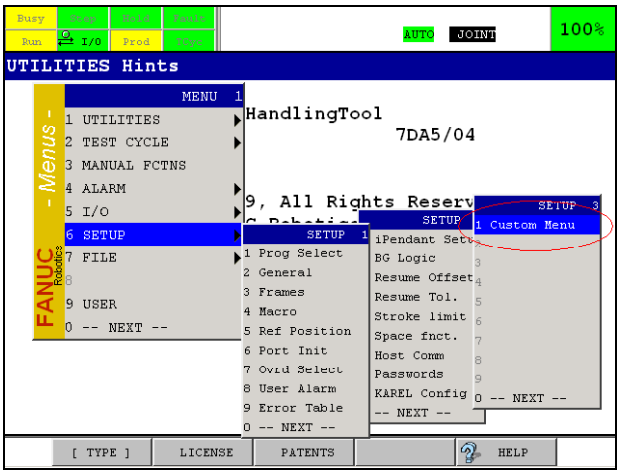


Selecting F4[YES] do cycle power.

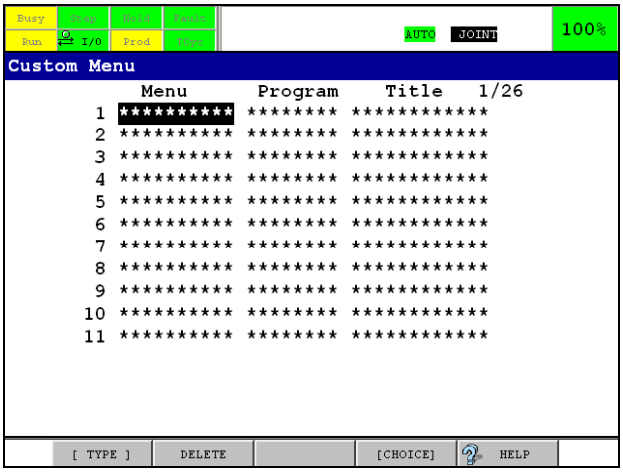
10.3 CUSTOM MENU

10.3.1 Starting Custom Menu

Select menu – setup – Custom Menu.

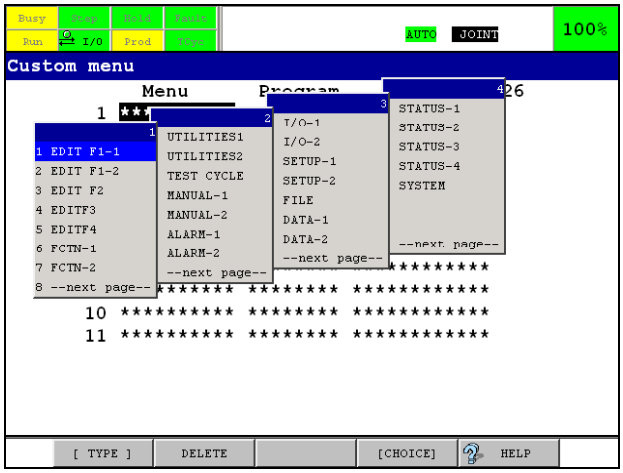


Following is displayed.

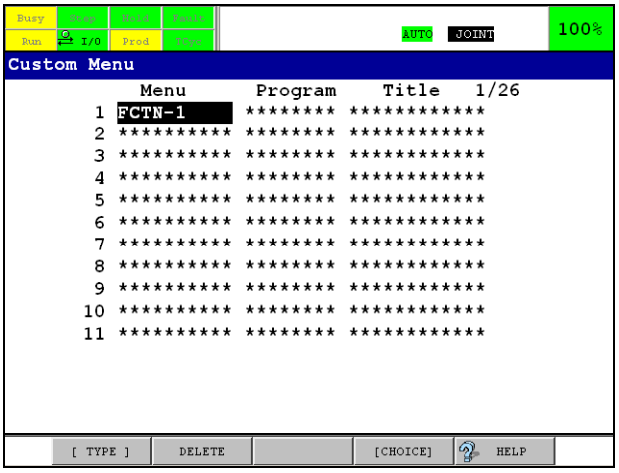


10.3.2 Set Custom Menu

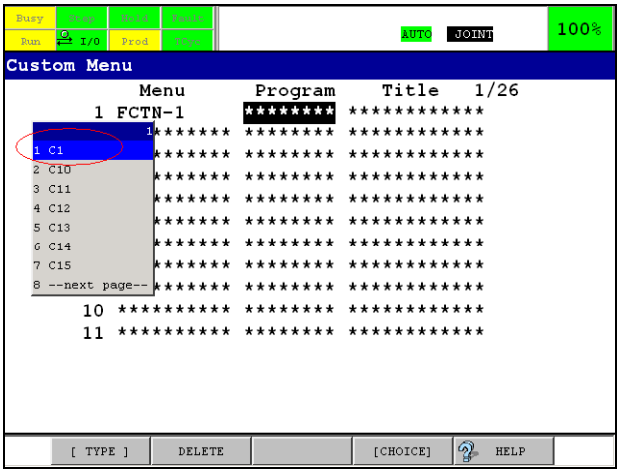
Press F4[CHOICE].



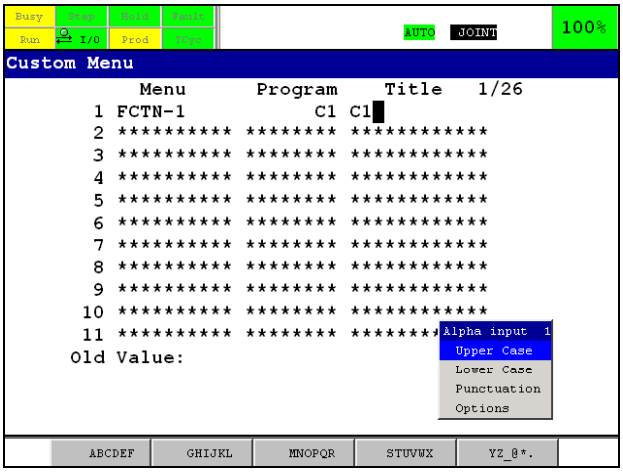
Select item that you would like to add. Select FCTN –1 as an example.



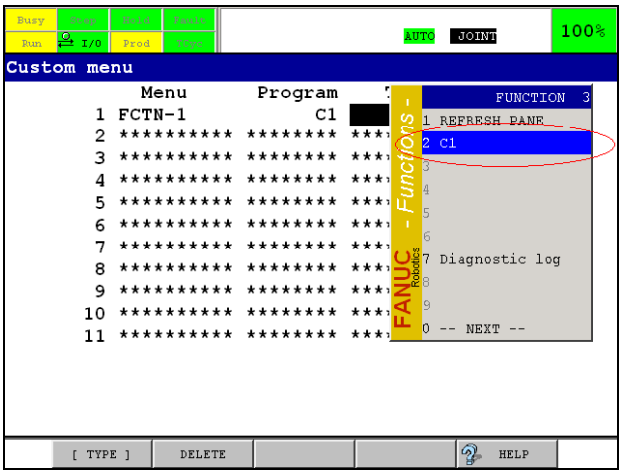
Set the cursor onto program row and press F4[CHOICE] key. As an example C1 is selected.



Next, set the cursor onto Title row and press ENTER key. C1 is input for instance.

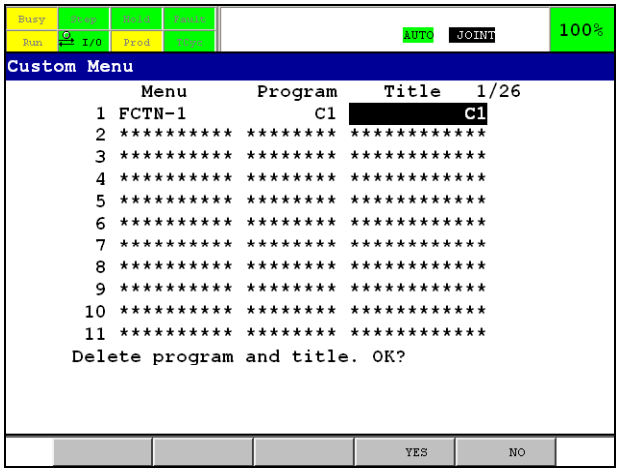


Select FCTN -> -- 0-- NEXT -> 0 –NEXT -- . Verify C1 is added to FCTN menu.

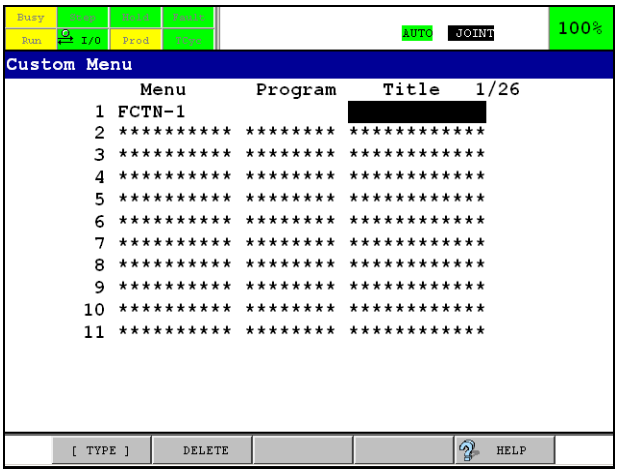


10.3.3 Delete Set

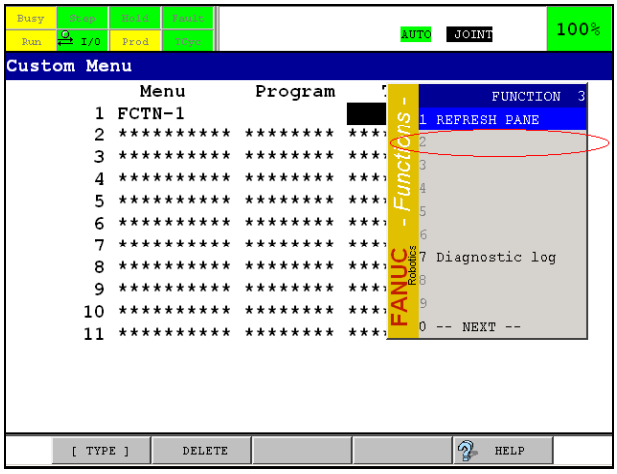
In the case of deleting set, press F2[DELETE]. Confirmation message is displayed like following.



Select F4[YES].
Specified program and title are deleted.



Select FCTN -> -- 0 --NEXT --> -- 0 --NEXT -- . Verify C1 is deleted from FCTN menu.



11 KAREL PROGRAM EXECUTION HISTORY RECORD

11.1 OVERVIEW

KAREL program execution history record is a tool used to diagnose user application or controller system software problems. This option generates logs that show details about events, including the sequence in which they occur.

It is allowed to log certain events that occur during KAREL and TP program execution. It is also allowed to generate ASCII files that contain all or part of the logged data. The ASCII files contain detail information about each event that is logged.

For example, it is allowed to log the following types of events.

- Start of execution of KAREL statements or TPP lines.
- Routine calls or returns.
- Internal system events for diagnosis of system software problems.

Refer to the following for additional information.

- For information about related hardware and software, refer to 11.2 Hardware And Software.
- For information about installing and setting up the debugging option, refer to 11.3 Setup And Operations.
- For information about events, refer to chapter 11.4 logging events.
- For examples of this option as related to KAREL programming, teach pendant programming, and the ASCII file format, refer to 11.5.

11.2 HARDWARE AND SOFTWARE

11.2.1 Hardware and Software Requirements

There are certain hardware and software requirements that must be met before you can run the KAREL Program Execution History Record.

NOTE
The hardware requirements in this section apply to all robot models.

11.2.1.1 Hardware

The KAREL Program Execution History Record requires the following hardware for all robot models:

- A FANUC Robot Controller.
- A teach pendant
- 42,000 bytes of Temporary memory
- 18,000 bytes of CMOS
- 30000 bytes of FROM file system space

11.2.1.2 Software

The KAREL Program Execution History Recode Option can be installed in controllers with KAREL software version 7DA5 or later.

A series of teach pendant is used to control the logging of diagnostic information. The teach pendant screens allow you to:

- Specify the kinds of events you want to log.
- Specify the tasks for which events are to be logged
- Generate ASCII files contain the logged data

11.2.2 Performance

The KAREL Program Execution History Record Option can affect the performance of your system.

- This option reduces execution speed of KAREL and teach pendant program logic by approximately 1 to 2 %.

NOTE
It is allowed to select “Disable all logging” on the Diagnostic Logging main menu to eliminate this reduction.

Refer to Subsection 11.3.8 for more information about system performance.

- Actual logging of extents adds about .08 ms per event logged.
- Motion execution is not affected.

11.3 SETUP AND OPERATIONS

11.3.1 Setting Up The KAREL Program Execution History Record

Before it allowed logging information and writing data log files, you must set up the KAREL Program Execution History Record Option. This section contains information about how to set up the KAREL Program Execution History Record Option, including a detailed procedure for each setup screen.

The following screen are used to control diagnostic login:

- "Clear event log" deletes all data from the log.
- "Dump Selections screen" is allowed to write log data to an ASCII file, Subsection11.3.2.
- "Task Selection screen" is allowed to add tasks for which events are logged, Subsection11.3.3.
- "Stop Logging a task Selection screen" is allowed to select tasks to be removed from the list of currently logged tasks, Subsection11.3.4.
- "Currently Selected tasks screen" lists tasks currently selected for logging, Subsection11.3.5.
- "Event Class Selections screen" is allowed to select or de-select classes or groups of related event types for logging, Subsection 11.3.6.
- "Event Detail selections screen" is allowed to select or de-select individual event types for logging, Subsection 11.3.7.
- "Enable or Disable all logging" is allowed to start or stop all event logging, Subsection 11.3.8.

For example, you can log the following types of events.

- Start of execution of KAREL statements or TPP lines.
- Routine calls or returns.
- Internal system events for diagnosis of system software problems.

11.3.2 Dump Selections Screen

The Dump Selection Screen is allowed to select a range of records to be dumped or logged. If the log is currently empty, an error message is displayed. Use Procedure 11-1 to dump records to the debug log.

Procedure 11-1 Dumping Records to the Debug Log

Steps

1. Press MENUS.
2. Select TEST CYCLE.
3. Press F1, [TYPE].
4. Select Debug Ctl, You will see a screen similar to the following.

```
DIAGNOSTIC LOGGING
  Main menu
    Clear event log
    Dump log data
    Add a task to log
    Stop logging a task
    List selected tasks
    Set classes to log
    Change events to log
    Enable all logging
    Disable all logging
```

5. Select Dump log data. The number of records in each range is displayed on the screen. See the following screen for an example.

```
DIAGNOSTIC LOGGING
 Dump Selections
    All log data
        (Will write 500 records)
    Since last power-up
        (Will write     1 records)
    Before last power-up
        (Will write 300 records)
    Reset file number
    Select RD: for files
    Select FLPY: for files
    Select MC: for files
```

6. Select the range of data to be dumped or logged.
   - To request a dump of all event information currently in the log , select All log data.
   - To request a dump of all event information logged since the most recent power-up, select Since last power-up.
   - To request a dump of all event information between the last two power-ups, select Before last power-up. This will log all events prior to the last power-down.

   The information will be dumped to an ASCII file.

NOTE
By default, successive log files are written to MC:PGDBG201.DT, MC:PGDBG202.DT, and so forth.

7. To reset the file number in the file naming convention , select Reset file number. The starting point is MC:PGDBG201.DT.
8. To change the default saving device to RAM disk, select RD: for files. All files will be automatically saved to RAM disk.
9. To change the default saving device to floppy disk, select FLPY: for files. All files will automatically be saved to floppy disk.
10. To change the default saving device to memory card, select MC: for files. All files will automatically be saved to a memory card.

11.3.3 Task Selection Screen

The Task Selection screen is allowed to select additional tasks for which you want to log information. Use Procedure 11-2 to select tasks to log.

Procedure 11-2 Selecting Tasks to Log

Steps

1. Press MENUS.
2. Select TEST CYCLE.
3. Press F1, [TYPE].
4. Select Debug Ctl. You will see a screen similar to the following.

```
DIAGNOSTIC LOGGING
  Main Menu
Clear event log
Dump log data

Stop logging a task
List selected tasks
Set classes to log
Change events to log
Enable all logging
Disable all logging
```

5. Select Add a task to log. You will see a screen similar to the following.

```
DIAGNOSTIC LOGGING
Task Selection                    1/1

Use CHOICE to select task to add, or EXIT
```

6. To display a list of tasks from which you can select, press F4, [CHOICE]. You will see a screen similar to the following.

NOTE

Selecting a program means that events will be logged for a task that has the selected program as its MAIN program.

```
1 PRG001                5 SUB0012
2 PRG002                6 SUB0013
3 PRG003                7 SUB0021
4 SUB0011
DIAGNOSTIC LOGGING
Task selection
Use CHOICE to select task to add, or EXIT
```

7. Use the arrow keys to select the task for which you want events logged.
8. Press ENTER when you have chosen a task. You will see a screen similar to the following.

```
DIAGNOSTIC LOGGING
Task selection                          1/1
Use CHOICE to select task to add, or EXIT
                PROG003
```

9. To request logging for the displayed task, press F3, OK.

⚠CAUTION
Do not exit this screen by pressing PREV, F1, [TYPE], or any other key before you press F3, OK; otherwise, the selected task will not be logged.

11.3.4 Stop Logging Tasks Screen

The Stop Logging Tasks screen is allowed to select tasks to be removed from the list of currently logged tasks. Use Procedure 11-3 to stop logging tasks.

Procedure 11-3 Stop Logging Tasks

Steps

1. Press MENUS.
2. Select TEST CYCLE.
3. Press F1, [TYPE].
4. Select Debug Ctl. You will see a screen similar to the following.

```
DIAGNOSTIC LOGGING
Main Menu
Clear event log
Dump log data
Add a task to log
Stop logging a task
List selected tasks
Set classes to log
Change events to log
Enable all logging
Disable all logging
```

5. Select Stop logging a task. You will see a screen similar to the following.

```
DIAGNOSTIC LOGGING
  Stop Logging
Use CHOICE to select task not to log
   or EXIT if none
```

6. To display a list of tasks, from which you can select, press F4[CHOICE]. You will see a screen similar to the following.

NOTE
When you select a program, the events will no longer be logged for any tasks that have the selected program as their MAIN program.

```
1 PRG001                5 SUB0012
2 PRG002                6 SUB0013
3 PRG003                7 SUB0021
4 SUB0011
DIAGNOSTIC LOGGING
Stop logging
```

7. Use the arrow keys to select the task for which you want to stop logging events.
8. Press ENTER when you have chosen a task. You will see a screen similar to the following.

```
DIAGNOSTIC LOGGING
Stop Logging                    1/1
Use CHOICE to select task not to log
or EXIT if none
PROG003
```

9. To stop logging for the task that is current displayed, press F3, OK.

⚠CAUTION
Do not exit this screen by pressing PREV, F1, [TYPE] , or any other key before you press F3, OK; otherwise, logging for the selected task will not stop.

11.3.5 List Selected Tasks Screen

The List Selected Tasks Screen is allowed to displays the tasks that are currently selected and being logged. Use Procedure 11-4 to list selected tasks.

Procedure 11-4 List Selected Tasks

Steps

1. Press MENUS.
2. Select TEST CYCLE.
3. Press F1, [TYPE].
4. Select Debug Ctl. You will see a screen similar to the following.

```
DIAGNOSTIC LOGGING
Main Menu
Clear event log
Dump log data
Add a task to log
Stop logging a task
List selected tasks
Set classes to log
Change events to log
Enable all logging
Disable all logging
```

5. Select List selected tasks. You will see a screen similar to the following.

```
DIAGNOSTIC LOGGING
Currently Selected Tasks
PRG003
PRG001
```

This screen displays the tasks that are currently selected and being logged.

6. Press PREV on the teach pendant to return to the main menu.

11.3.6 Event Class Selection Screen

The Event Class Selection Screen is allowed to determine which classes of events should be logged (Procedure 11-5). Each class of events enables or disables one or more detailed event types. The following is a list of event classes and the associated event types. Refer to Subsection 11.4.2 for more detailed information on the types of events that can be logged.

NOTE
An asterisk (*) indicates logging is for internal use only.

- Execute KAREL statement
  - Execute KAREL statement
- Execute TPP line
  - Execute TPP line
- Call/return
  - KAREL or TPP routine called
  - KAREL or TPP routine returned
- Motion
  - Motion started
  - Motion planned (teach pendant motion only)
  - Motion cancel issued
  - Motion stop issued
  - Resume move
  - Motion done received
  - Motion completed normally
  - MMR received*
- Condition Handler
  - Condition handler triggered
  - Condition handler enabled
  - Condition handler disabled
- Interrupt rtn
  - Before transferring to an ISR (Interrupt Sub-Routine)
  - After transferring to an ISR (Interrupt Sub-Routine)
  - Return from ISR (Interrupt Sub-Routine)
- Task start/end
  - KAREL or TPP task starts execution
  - KAREL or TPP task aborts
- Packet_rcd *
  - Interpreter receives packet
- Pcode exec *
  - Starting execution of p-code
- AMR activity *
  - Normal AMR recvd by AMGR
  - Start AMR recvd by AMGR
  - Stop AMR recvd by AMGR

- AMR sent to AX
- AMR rcvd from AX

Use Procedure 11-5 to enable or disable event classes for logging.

Procedure 11-5 Enable or Disable Event Classes for Logging

Steps

1. Press MENUS.
2. Select TEST CYCLE.
3. Press F1, [TYPE].
4. Select Debug Ctl. You will see a screen similar to the following.

```
DIAGNOSTIC LOGGING
Main Menu
Clear event log
Dump log data
Add a task to log
Stop logging a task
List selected tasks
Set classes to log
Change events to log
Enable all logging
Disable all logging
```

5. Select Set classes to log. You will see a screen similar to the following.

```
DIAGNOSTIC LOGGING
Event Class Selections                1/10
   NO   Execute KAREL line
   NO   Execute TPP line
   NO   Call/return
   NO   Motion
   NO   Condition Handler
   NO   Interrupt rtn
   NO   Task start/end
   NO   Packet_rcd
   NO   Pcode exec
```

6. Enable or disable classes of events to be logged.

- To turn on logging for a class of events , press F4, YES.

- To turn off logging for a class of events , press F5, NO.

The changes take effect immediately. The current status of whether an event is being logged is displayed to the left of the event.

NOTE

If you specify an entire class of events, you will enable or disable logging for all event types in that class.

11.3.7 Event Detail Selection Screen

The Event Detail Selections Screen is allowed to enable and disable logging of specific events. Use Procedure 11-6 to enable or disable a specific event type for logging.

Procedure 11-6 Enable or Disable a Specific Event Type for Logging

Steps

1. Press MENUS.
2. Select TEST CYCLE.
3. Press F1, [TYPE].
4. Select Debug Ctl. You will see a screen similar to the following.

```
DIAGNOSTIC LOGGING
Main Menu
Clear event log
Dump log data
Add a task to log
Stop logging a task
List selected tasks
Set classes to log
Change events to log
Enable all logging
Disable all logging
```

5. Select Change events to log. You will see a screen similar to the following.

```
DIAGNOSTIC LOGGING
Event Detail Selections                1/30
  YES   Controller power-up
  YES   Logging enabled
  YES   Logging disabled
   NO   Execute KAREL line
   NO   KAREL or TPP routine called
   NO   KAREL or TPP routine returned
  YES   Motion started
  YES   Motion planned
  YES   Motion cancel issued
```

6. Enable or disable a specific event type to be logged.

- To turn on logging for an event type, press F4, YES.
- To turn off logging for an event type, press F5, NO.

The changes take effect immediately.

11.3.8 Enable or Disable All Event Logging

The Enable/Disable All Logging Screen is allowed to start or stop the automatic logging of all selected events. When the Program Diagnostic Option is installed, logging of selected events in selected tasks is enabled. The Disable all logging menu selection turns off all logging. It also eliminates the overhead added when logging is enabled (refer to Subsection 11.2 "Performance.") The Enable all logging menu selection can be used to start logging again.

NOTE
Event and task selections are not changed by Disable all logging *or* Enable all logging .

Use Procedure 11-7 to enable and disable all event logging.

Procedure 11-7 Enabling and Disabling All Event Logging

Steps

1. Press MENUS.
2. Select TEST CYCLE.
3. Press F1, [TYPE].
4. Select Debug Ctl. You will see a screen similar to the following.

DIAGNOSTIC LOGGING
Main Menu
Clear event log
Dump log data
Add a task to log
Stop logging a task
List selected tasks
Set classes to log
Change events to log
Enable all logging
Disable all logging

5. Select either Enable or Disable all logging.

⚠CAUTION
If you Enable or Disable all logging, you must perform a Cold Start for the change to take effect.

- If you select Enable all logging, selected events in selected tasks will automatically start being logged after you perform a Cold Start.
- If you select Disable all logging, selected events in selected tasks will automatically stop being logged after you perform a Cold Start.

6. Execute a Cold Start.

Conditions

- All personnel and unnecessary equipment are out of the work cell.

⚠WARNING
DO NOT turn on the robot if you discover any problems or potential hazards. Report them immediately. Turning on a robot that does not pass inspection could result in serious injury.

a. Visually inspect the robot, controller, work cell, and the surrounding area. During the inspection make sure all safeguards are in place and the work envelope is clear of personnel.
b. Turn the power disconnect circuit breaker to ON.
c. When the BMON screen is displayed on the teach pendant, press and hold the SHIFT and RESET keys. Or, on the operator panel, press and hold RESET.
d. Release all of the keys.

- On the operator panel the ON LED will be illuminated, indicating robot power is ON.
- On the teach pendant screen, you will see a screen similar to the following.

```
UTILITIES Hints                    JOINT 100%
                ApplicationTool
          Vx.xxx                XXXX/XX
                Custom Vx.xxx
  Copyright xxxx, All Rights Reserved
FANUC LTD, FANUC Robotics America, Inc.
Licensed Software: Your use constitutes
your acceptance. This product protected
by several U.S. patents.
```

11.4 LOGGING EVENTS

Events are conditions or situations that occur in a KAREL program or TP program while the program is running. The KAREL Program Execution History Record Option is allowed to record certain events, as they take place, to help you in debugging programs. The logged events can be dumped to an ASCII file.

11.4.1 Setting Up Events

An event will be recorded in the log when all of the following are true:

- You have selected Enable all logging from the main menu.
- You have enabled event logging for the task.
- You have selected the event type to log either,
  - Specifically, using the Change Events To Log screen to select a specific event.
  - Generally, using the Set Classes To Log screen to select a class of events.

Types of Events that can be Logged

With the Program Diagnostic Option, the following kinds of events can be recorded. Refer to Section 4.2.2 for additional information.

- Controller power-up
- Logging enabled
- Logging disabled
- Execute KAREL line
- KAREL or TPP routine called
- KAREL or TPP routine returned
- Motion started
- Motion planned
- Motion cancel issued
- Motion stop issued
- Resume move
- Motion done received
- Motion completed normally
- Condition handler triggered
- Condition handler enabled
- Condition handler disabled
- Before processing ISR
- Return from KAREL ISR
- KAREL or TPP task starts execution
- KAREL or TPP task aborts
- Interpreter receives packet

- Starting execution of p-code
- Start of execution of TPP line
- MMR_RCVD (MMR received)
- MMR received
- Normal AMR received by AMGR
- Start AMR received by AMGR
- Stop AMR received by AMGR
- AMR sent to AX
- AMR received from AX

11.4.2 Logging Events to An ASCII File

The last 500 events that are recorded can be written to an ASCII file. This can be done by requesting to write the logged data to a file on the Dump Selections screen (refer to Subsection 11.3.2).

11.4.3 ASCII File General Event Information

Specific information about each event is logged to the ASCII file. Table11.4.3.1.describes the following information for each event:

- Event name: Event description displayed in the Event Detail Selections screen.
- When recorded: Conditions under which the event is recorded. For example, the STRT_K_LINE event is recorded when a KAREL statement is about to be executed.
- Information recorded: List of values logged with each event. The following are reported for all events (except as noted).
  - Event time (seconds) since power-up. This is always a multiple of 4 milliseconds (.004 second).
  - Number of the task associated with the event.
  - Name of the KAREL routine or TPP program, in which the statement was triggered.
  - Program line number at which statement event was triggered.
- Enabled default: YES if logging of the specified event type is Enabled by default.
- Comments: Additional information, if appropriate.

11.4.3.1 ASCII file specific event information

Table11.4.3.1 contains each event and the corresponding information that is logged.

Table11.4.3.1. Event Logging Information

Event Name	When Recorded	Information Recorded	Enabled Default	Comments
Controller power-up	At every controller COLD start	Routine name,line number		
		Clock time: at which power-up occurred, in "DD-MMM-YY HH:MM" format.		
		Task number will always be 1.		
Logging enabled	When logging is started. Typically, this is when the task starts.	Standard information only	YES	
Logging disabled	When logging is stopped. Typically, this is when the task ends.	Standard information only	YES	

Event Name	When Recorded	Information Recorded	Enabled Default	Comments
Execute KAREL line	At beginning of execution of KAREL statement.	Internal information	NO	
		Internal information		
		Internal information		
KAREL or TPP routine called	When a KAREL routine or KAREL or TPP program is called. This is either directly or as a result of a condition handler action.	Standard information only	NO	Routine name is name of called routine; line number is line number from which call was made or interrupt occurred.
KAREL or TPP routine returned	When a KAREL PROGRAM or ROUTINE or TP program returns to the calling or interrupted program	Standard information only	NO	The ROUTINE name is the name of the returning program or routine; the line number is the line in the calling or interrupted program to which the task is returning, generally the same as the line number shown for in the CALL event.
Motion started	When a request to initiate a motion is issued.	Group mask	NO	Intended for system level analysis. .
		MMR address, in hexadecimal		
		MMR status, in hexadecimal; normally FFFFFFFF		
Motion planned	For TPP motion commands, this occurs typically before the motion is actually started.	Group mask	NO	Intended for system level analysis. .
		MMR address, in hexadecimal		
		MMR status, in hexadecimal; normally FFFFFFFF		
Motion cancel issued	When a CANCEL statement or condition handler action is executed, a MOVE... UNTIL condition is satisfied, a CANCEL severity error is posted, or a program is aborted with a motion planned or in progress.	Group mask	NO	Intended for system level analysis. .
		MMR address, in hexadecimal		
		MMR status, in hexadecimal; normally FFFFFFFF		

Event Name	When Recorded	Information Recorded	Enabled Default	Comments
Motion stop issued	When a STOP statement or condition handler action is executed or a STOP severity error is posted.	Group mask MMR address, in hexadecimal MMR status, in hexadecimal; normally FFFFFFFF	NO	Intended for system level analysis. .
Resume move	When a stopped motion is resumed; typically when a RESUME statement or condition handler action is executed; also when a program is CONTINUED following a STOP error condition.	Group mask MMR address, in hexadecimal MMR status, in hexadecimal; normally FFFFFFFF	NO	Intended for system level analysis. .
Motion done received	When the termination type is satisfied for a KAREL or TPP motion statement is executed. For KAREL MOVE ... NOWAIT statements, this event if recorded when the motion starts.	Group mask MMR address, in hexadecimal MMR status, in hexadecimal; normally FFFFFFFF	NO	Intended for system level analysis. .
Motion completed normally	When a motion is completed or cancelled. This is generally after MTN_DONE is recorded.	Group mask MMR address, in hexadecimal MMR status, in hexadecimal; normally FFFFFFFF		Intended for system level analysis. .
Condition handler triggered	When a global condition handler triggers	Intended for system level analysis.	NO	Intended for system level analysis. .
Condition handler enabled	When a global condition handler is enabled by a ENABLE statement or condition handler action.	Condition handler number	NO	Intended for system level analysis. .
Condition handler disabled	When a global condition handler is disabled by a DISABLE statement or condition handler action. It is not recorded when a condition handler triggers.	Condition handler number	NO	Intended for system level analysis. .
Before processing	Before calling a KAREL PROGRAM	Internal information Internal information	NO	The routine name and line number indicate the

Event Name	When Recorded	Information Recorded	Enabled Default	Comments
ISR	from a condition handler.	Internal information		code that was executing when the interrupt routine request was received.
After processing ISR	When the interrupt routine is ready to run.	Standard information only	NO	Routine name is the name of the interrupt routine. Line number will always be 1.
Return from KAREL ISR	When exiting from an interrupt routine.	Standard information only	NO	Routine name and line number are the interrupt routine name and the line number from which it returned.
KAREL or TPP task starts execution	When a task selected for logging starts executing.	Standard information only	YES	Routine name is the main program name; line number is always 1.
KAREL or TPP task aborts	When a task ends.	Standard information only	YES	Routine and line number indicate the last statement executed by the task. This might be an END statement of a KAREL program or the last line of a TPP program.
Interpreter receives packet	When an interpreter task receives a packet.	Packet address	NO	Intended for system level analysis.
		Packet status		
		Request code (including sub-system code)		
		Requestor id		
		ITR-level		
Starting execution of p-code	At start of execution of KAREL p-code instruction.	Number of words on the routine stack	NO	Intended for system level analysis.
		Number of words on the data stack		
		P-code mnemonic		
Start of execution of TPP line	At start of execution of a line of a TPP program.	Number of words on the data stack	NO	
		Number of words on the routine stack		
		Number of words available for combined ROUTINE and DATA stacks		

Event Name	When Recorded	Information Recorded	Enabled Default	Comments
MMR received	When an MMR is received back from the motion sub-system. This occurs when the motion completes, is cancelled, or is stopped.	Group mask	NO	
		MMR address, in hexadecimal		
		MMR status, in hexadecimal; normally FFFFFFFF		
Normal AMR recvd by AMGR	An AMR is received by AMGR.	Address of AMR	NO	Intended for system level analysis.
		AMR number		
		AMR AMGR_wk		
		AMR ax_phase		
Start AMR recvd by AMGR	A start AMR request is processed by AMGR.	Address of AMR	NO	Intended for system level analysis.
		AMR number		
		AMR AMGR_wk		
		AMR ax_phase		
Stop AMR recvd by AMGR	A stop AMR request is processed by AMGR.	Address of AMR	NO	Intended for system level analysis.
		AMR number		
		AMR AMGR_wk		
		AMR ax_phase		
AMR sent to AX	When an AMR is sent from AMGR to an AX task.	Address of AMR	NO	Intended for system level analysis.
		AMR number		
		AMR AMGR_wk		
		AMR ax_phase		
AMR rcvd from AX	When an AMR is receive back from an AX task.	Address of AMR	NO	Intended for system level analysis.
		AMR number		
		AMR AMGR_wk		
		AMR ax_phase		

11.5 EXAMPLES

11.5.1 Overview

This appendix contains example programs and the resulting log data, that show how the KAREL Program Execution History Record Option operates.

- Subsection 11.5.2 contains a KAREL program (T) which calls a teach pendant program (TPP).
- Subsection 11.5.3 contains the teach pendant program that is called (TTT).
- Subsection 11.5.4 contains the log file that is generated after running the KAREL program (T) with the following Event class selections:
  - YES Execute KAREL line
  - YES Execute TPP line
  - YES Call/return
  - YES Motion

- YES Condition Handler
- YES Interrupt rtn
- YES Task start/end
- NO Packet_rcd
- NO Pcode exec

11.5.2 KAREL Program Example

The KAREL program in Example 11.5.2 is the MAIN program from which the TPP program is called and executed.

Example 11.5.2 KAREL Program Example (T.KL)

```
 1 program t
 2 var
 3     p1,p2: xyzwpr
 4     i,j,k: INTEGER
 5
 6 routine ttt from ttt
 7 routine tt
 8 begin
 9     i = i + 1
10 end tt
11
12 begin
13 if uninit(p1) THEN
14     p1 = curpos(0,0)
15     p2 = p1
16     p2.x = p2.x + 100
17 endif
18 condition[1]:
19     when k = 500 DO
20         k = 0
21         tt
22         enable condition[1]
23     endcondition
24 k = 0
25 i = 0
26 connect timer to k
27 enable condition[1]
28 wait for i=2
29 disconnect timer k
30 disable condition[1]
31 move to p1
32 WITH $SPEED=50 move to p2
33 ttt
34 delay 1000
35 end t
```

11.5.3 Teach Pendant Program Example

The TPP program in Example 11.5.3 is called by the KAREL program (T) in Example 11.5.2.

Example 11.5.3 Teach Pendant Program Example (TTT.TP)

```
  1:  J P[1] 100% FINE
  2:  WAIT     0.00(sec)
  3:  R[1]=0
  4:  LBL[1]
  5:  R[1]=R[1]+1
  6:  IF R[1] <= 3,JMP LBL[1]
  7:  WAIT      .50(sec)
```

11.5.4 ASCII File Example

With logging task T enable and event classes selected as indicated in Example 11.5.2 , if T is executed and the log dumped, the file show in Example 11.5.4 is generated.

NOTE
For detail information about the fields on the ASCII report shown in Example11.5.4, refer to Subsection 11.4.3.1.

Example 11.5.4 ASCII File Example

```
Event        Time TID        Routine Line
     POWER-UP      10.588    1              ***   *** 02-JUN-95 09:31
   TASK-START     159.668    2                T      1
   LOG-ENABLE     159.668    2                T      1
  STRT-K-LINE     159.668    2                T     13      0      0   300
  STRT-K-LINE     159.680    2                T     14      0      0   300
  STRT-K-LINE     159.684    2                T     15      0      0   300
  STRT-K-LINE     159.684    2                T     16      0      0   300
  STRT-K-LINE     159.684    2                T     17      0      0   300
  STRT-K-LINE     159.684    2                T     18      0      0   300
  STRT-K-LINE     159.688    2                T     24      0      0   300
  STRT-K-LINE     159.688    2                T     25      0      0   300
  STRT-K-LINE     159.688    2                T     26      0      0   300
  STRT-K-LINE     159.688    2                T     27      0      0   300
  STRT-K-LINE     159.692    2                T     28      0      0   300
    CH-ENABLE     159.688    2                T     28      1
   CH-TRIGGER     160.176    2                T     28      1
      PRE-ISR     160.176    2                T     28      0    128     0
         CALL     160.176    2               TT     28
       IN-ISR     160.176    2               TT      1
  STRT-K-LINE     160.176    2                TT     9      0     50   250
  STRT-K-LINE     160.184    2                TT    10      0     50   250
       RETURN     160.184    2                TT     28
  RTN-INT-RTN     160.184    2                TT     28
   CH-TRIGGER     160.676    2                T     28      1
      PRE-ISR     160.676    2                T     28      0    128     0
         CALL     160.676    2               TT     28
       IN-ISR     160.680    2               TT      1
  STRT-K-LINE     160.680    2                TT     9      0     50   250
```

```
STRT-K-LINE    160.688    2          TT    10     0    50   250
      RETURN    160.688    2          TT    28
RTN-INT-RTN    160.688    2          TT    28
 CH-TRIGGER    161.176    2           T    28     1
    PRE-ISR    161.176    2           T    28     0   128     0
       CALL    161.176    2          TT    28
     IN-ISR    161.176    2          TT     1
STRT-K-LINE    161.176    2          TT     9     0    50   250
STRT-K-LINE    161.184    2          TT    10     0    50   250
      RETURN    161.184    2          TT    28
RTN-INT-RTN    161.184    2          TT    28
STRT-K-LINE    161.204    2           T    29     0     0   300
STRT-K-LINE    161.204    2           T    30     0     0   300
STRT-K-LINE    161.208    2           T    31     0     0   300
 CH-DISABLE    161.208    2           T    31     1
 START-MOVE    161.208    2           T    31 018C5F1C FFFFFFFF     0
   MTN-DONE    161.260    2           T    31 018C5F1C FFFFFFFF     0
STRT-K-LINE    161.260    2           T    32     0     0   300
START-MOVE     161.260    2           T    32 018C5F70 FFFFFFFF   248
    MMR_RCVD   161.284    2           T    31 018C5F1C 00000000     0
  MTN_ENDED    161.284    2           T    31 018C5F1C 00000000     0
   MTN-DONE    162.196    2           T    32 018C5F70 FFFFFFFF     0
STRT-K-LINE    162.196    2           T    33     0     0   300
       CALL    162.204    2          TTT    33
STRT-T-LINE    162.204    2          TTT     1     0    10   290
  PLAN-MOVE    162.204    2           TTT     1 018C5FC4 FFFFFFFF     0
 START-MOVE    162.204    2           TTT     1 018C5FC4 FFFFFFFF     0
    MMR_RCVD   162.220    2           T    32 018C5F70 00000000     0
  MTN_ENDED    162.224    2           T    32 018C5F70 00000000     0
   MTN-DONE    162.536    2           TTT     1 018C5FC4 FFFFFFFF     0
STRT-T-LINE    162.536    2          TTT     2     0    10   290
    MMR_RCVD   162.564    2           TTT     1 018C5FC4 00000000     0
  MTN_ENDED    162.564    2           TTT     1 018C5FC4 00000000     0
STRT-T-LINE    162.564    2          TTT     3     0    10   290
STRT-T-LINE    162.564    2          TTT     4     0    10   290
STRT-T-LINE    162.568    2          TTT     5     0    10   290
STRT-T-LINE    162.568    2          TTT     6     0    10   290
STRT-T-LINE    162.572    2          TTT     4     0    10   290
STRT-T-LINE    162.572    2          TTT     5     0    10   290
STRT-T-LINE    162.576    2          TTT     6     0    10   290
STRT-T-LINE    162.580    2          TTT     4     0    10   290
STRT-T-LINE    162.580    2          TTT     5     0    10   290
STRT-T-LINE    162.580    2          TTT     6     0    10   290
STRT-T-LINE    162.580    2          TTT     7     0    10   290
      RETURN    163.080    2          TTT    33
STRT-K-LINE    163.080    2           T    34     0     0   300
STRT-K-LINE    164.084    2           T    35     0     0   300
   TASK-END    164.108    2           T    35
LOG-DISABLE    164.108    2           T    35
```

12 DISPLAY ON PC

Connecting PC and the robot controller through Ethernet, web page of the robot controller is allowed to display on PC browser, Internet Explorer. KAREL program that has no group is allowed to execute from PC. JavaScript is allowed to use on the browser.

12.1 ADDRESS SPECIFIED FROM BROWSER (URL)

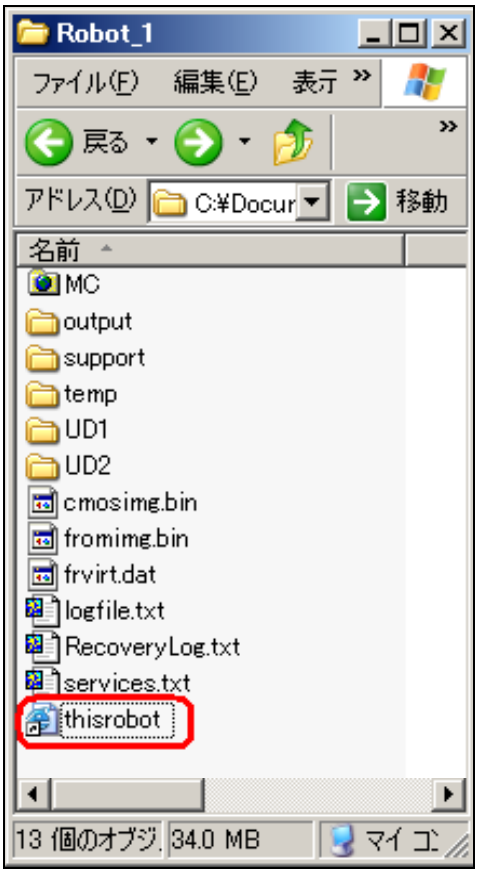
This section mainly explains URL setting. Refer to Ethernet Function OPERATOR'S MANUAL (B-82974EN/01), 6.3 USING THE WEB SERVER to see detail. Refer to chapter 2, SETTING UP TCP/IP of it to see the way to connect the robot and Ethernet. Usual syntax is following.

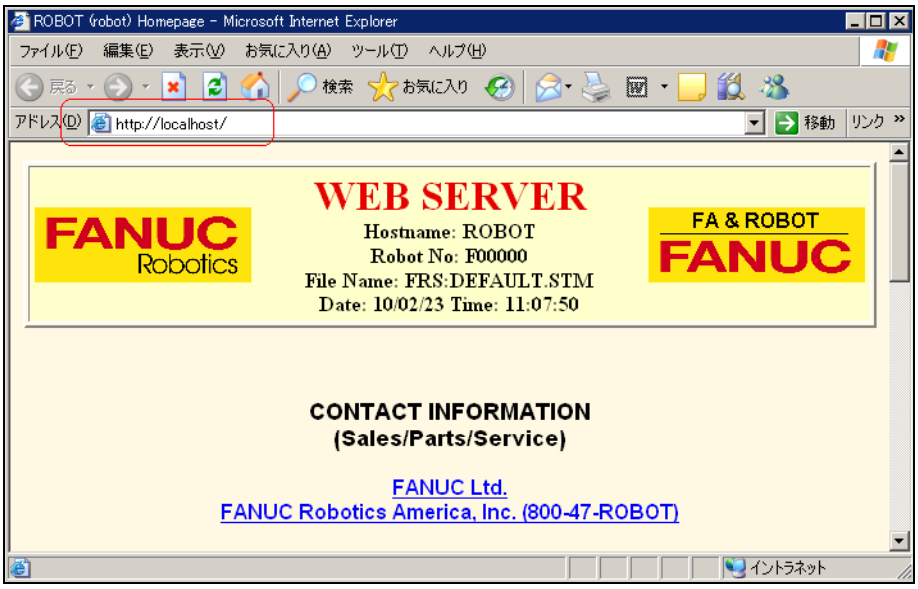
```
http://<robot>/<device>/<filename>
```

The part of "robot" is host name or IP address of the robot. "device" is device name. For example, in case that IP address is 192.168.0.1 and specifies test.htm of MC: root directory, describe as follows.

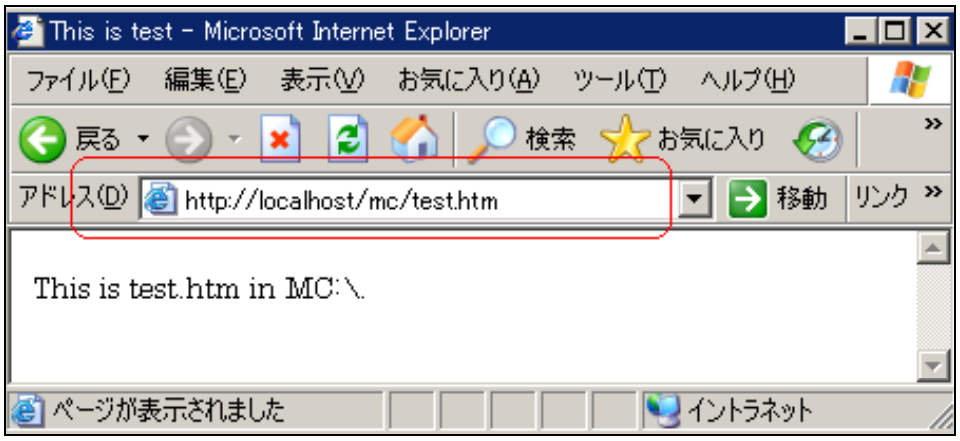
```
http://192.168.0.1/mc/test.htm
```

In case of virtual robot the short cut, this robot, in virtual robot directory is allowed to use. Do double click to display robot homepage.

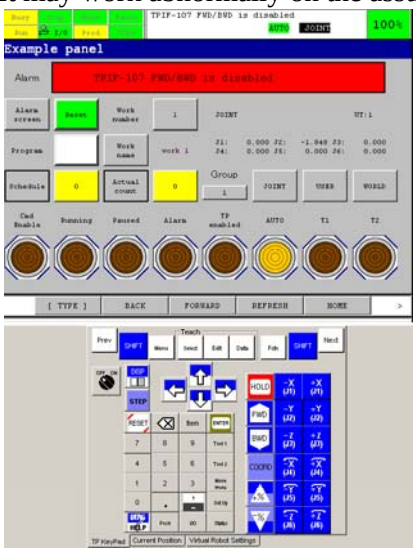




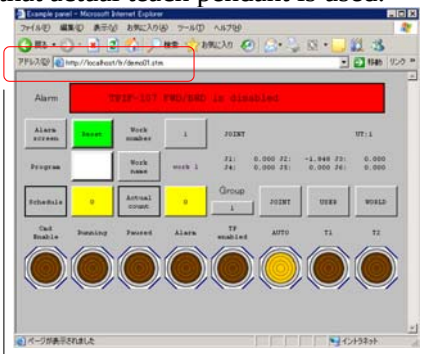
Corresponding part of robot IP address is already defined, and then input after device name. In above figure, if test.htm is existed in root directory of MC:, IP address is as follows.



A screen created by *i*Pendant Screen Customization Function is allowed to display if ROBOGUIDE or it is installed in PC. It may work abnormally on the assumption that actual teach pendant is used.



Display in *i*Pendant



Display in browser.

FR:¥demo01.stm is specified.

12.2 EXECUTE KAREL PROGRAM FROM PC

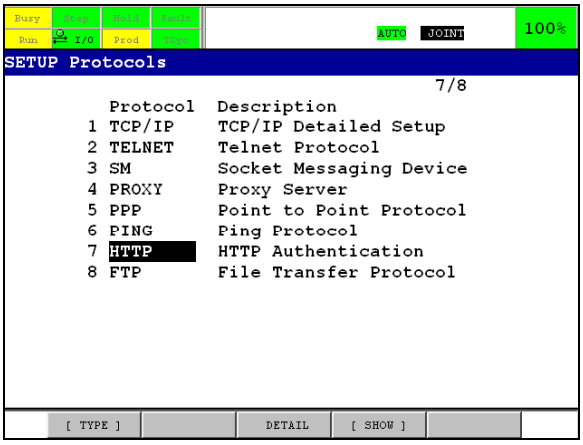
HTTP authentication is needed to execute KAREL programs from web server. In case of no necessity it is allowed to execute setting UNLOCK. Refer to Ethernet Function OPERATOR'S MANUAL (B-82974EN/01), 6.5 HTTP AUTHENTICATION about detail of it.

Procedure 12-1 Set UNLOCK to HTTP authentication of KAREL

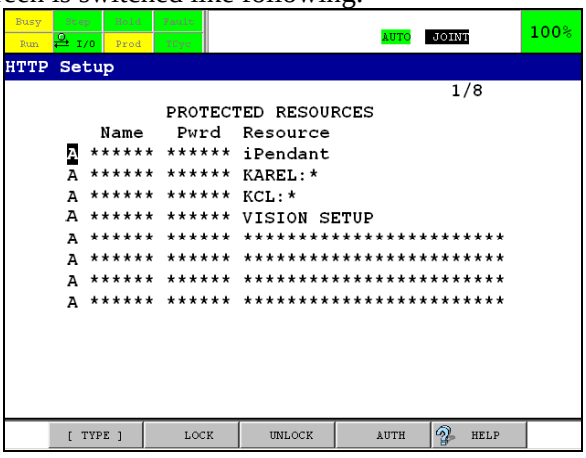
STEP

1 Select Menu -> 6 SETUP -> Host Comm.

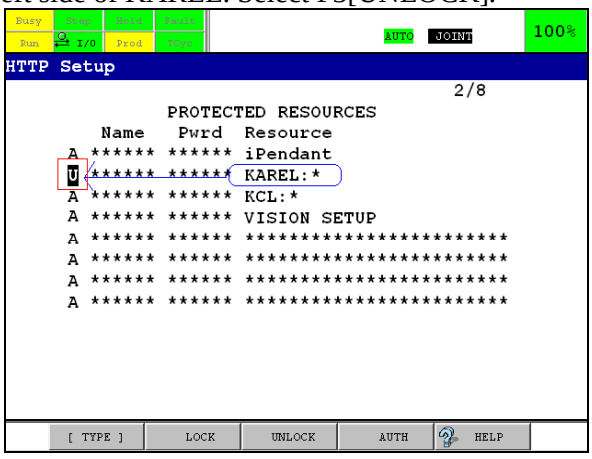
2 Set the cursor on HTTP.



3 Push F3[DETAIL]. Screen is switched like following.

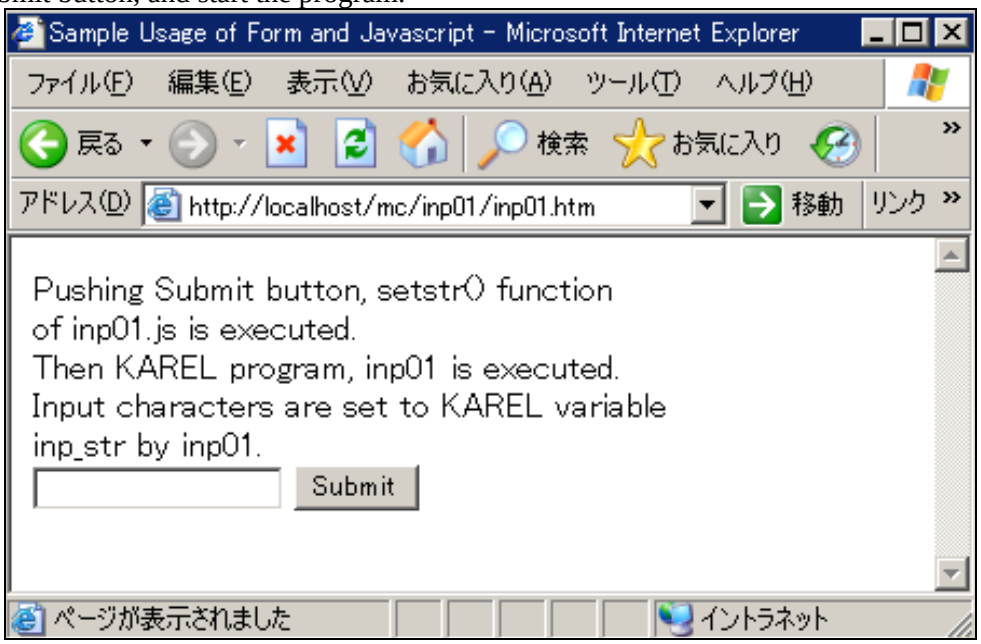


4 Set the cursor on "A" left side of KAREL. Select F3[UNLOCK].



12.3 FORM USAGE SAMPLE 1

This section explains a creating way of web page as follows. Input characters into text box on web page and set them by submit button, and start the program.



NOTE
It is the sample to display the web page on the browser (Internet Explorer) of PC connected the robot controller and Ethernet. It is not the page on *i*Pendant.

This page makes from three elements.

- /mc/inp01/inp01.htm :Web page
- /mc/inp01/inp01.js :JavaScript
- /mc/inp01/inp01.kl :KAREL program to set variables and start

In this sample KAREL starts by specified URL. Refer to Ethernet Function OPERATOR'S MANUAL (B-82974EN/01), 6.3.5 amd 6.3.6 to see detail.

inp01.htm is as follows.

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=shift_jis">
<script language="javascript" src="/mc/inp01/inp01.js"></script>
<title>Sample Usage of Form and JavaScript</title>
</head>
<body>
<form method="POST">

    <p>Pushing Submit button, setstr() function<br>
    of inp01.js is executed.<br>
    Then KAREL program, inp01 is executed.<br>
    Input characters are set to KAREL variable <br>
          inp_str by inp01.<br>
```

```
    <input type="text" name="T1" size="20">
    <input type="button" value="Submit" name="B1"
           onclick="JavaScript:setstr(T1.value)"></p>
</form>
</body>
</html>
```

JavaScript is specifier as script and source file is /mc/inp01/inp01.js. Function setstr() is called by onclick event of submit button. The value of text box T1 is given as second argument.

Next shows inp01.js.

```
//
//KAREL Interface file
//

function setstr(strInp) {
   document.location.href = '/KAREL/inp01?inp_str='+strInp
}
```

URL specifies execution of KAREL program, inp01. On this occasion, variable INP_STR received from Form characters is set to strInp. It should be STRING type variable to set.

Below is inp01.kl.

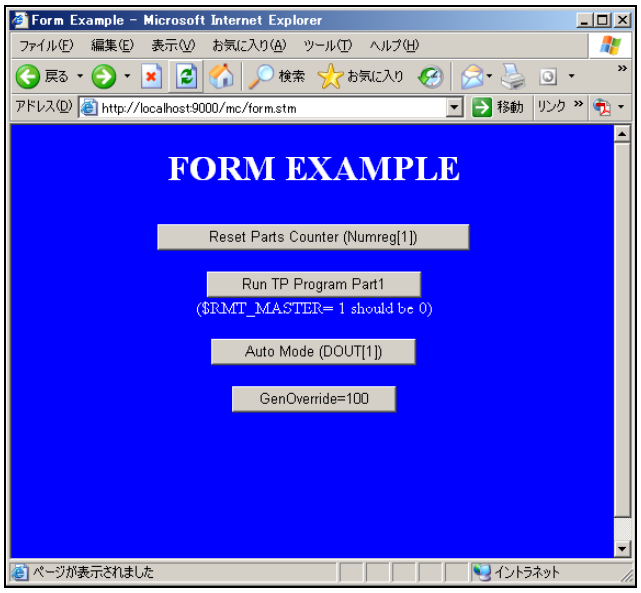
```
PROGRAM inp01

%NOLOCKGROUP

VAR
   inp_str : STRING[127]
   return_code                    : INTEGER
BEGIN
   return_code = 204
   WRITE (inp_str, CR)
END inp01
```

On startup inpstr is set by setstr(). It is described on USER screen. It does not create response file, variable return_code is set to 204.

12.4 FORM USAGE SAMPLE 2

12.4.1 Overview

This subsection explains following web page.



NOTE
It is the sample to display the web page on the browser (Internet Explorer) of PC connected the robot controller and Ethernet. It is not the page on *i*Pendant.

This page makes from two elements.

- /mc/form.stm :Web page
- npnlsvr.kl :KAREL program

The part of button, “Counter of parts reset”, is described in form.stm as follows.

```
<form action="../../Karel/mpnlsvr" method="GET">
  <div align="center">
    <input type="hidden" name="object" value="numreg">
    <input type="hidden" name="operate" value="setint">
    <input type="hidden" name="index" value="1">
    <input type="hidden" name="value" value="0">
    <input type="submit" value="Reset Parts Counter (Numreg[1])">
  </div>
</form>
```

KAREL program, mpnlsvr, is specified to execute in form action. The value of KAREL variable (only STRING) specified as name of INPUT tag of type=”hidden” is set. For example, KAREL variable object is set to numreg. KAREL program, mpnlsvr, is execute the motion corresponding to set variables. Following is the clip that the case object is NUMREG. “unobject” is upper size of object.

```
-- Treating numeric register setting
   if (uobject = 'NUMREG') then
         cnv_str_int(uindex, index_i)
      if (uoperate = 'SETINT') then
         cnv_str_int(uvalue, value_i)
         set_int_reg(index_i, value_i, status)
      endif

      if (uoperate = 'SETREAL') then
         cnv_str_real(uvalue, value_r)
         set_real_reg(index_i, value_r, status)
      endif
   endif
```

12.4.2 form.stm

```
<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<title>Form Example</title>
</head>

<body bgcolor="#0000FF">

<div align="left">
   <table border="0" cellpadding="0" cellspacing="0">
      <tr>
         <td valign="top" align="left" height="388" width="5" rowspan="2"><img
    border="0" src="spacer.gif" width="1" height="1"></td>
         <td valign="top" align="left" height="5" width="623"><img border="0"
    src="spacer.gif" width="1" height="1"></td>
      </tr>
      <tr>
         <td valign="top" align="center">
            <div align="center">
               <table width="99%" border="0" cellspacing="0" cellpadding="0"
    align="center" height="186">
                  <tr valign="top" align="center">
                     <td valign="top" height="21"><font size="6"
    color="#FFFFFF"><b>FORM
                        EXAMPLE</b></font></td>
                  </tr>
                  <tr>
                     <td valign="top" height="33">
                     </td>
                  </tr>
```

```
            <tr>
              <td height="30">
                  <form action="../../Karel/mpnlsvr" method="GET">
                    <div align="center">
                      <input type="hidden" name="object" value="numreg">
                      <input type="hidden" name="operate" value="setint">
                      <input type="hidden" name="index" value="1">
                      <input type="hidden" name="value" value="0">
                      <input type="submit" value="Reset Parts Counter
(Numreg[1])">
                    </div>
                  </form>
              </td>
            </tr>
            <tr>
                  <td height="30">
                    <form action="../../Karel/mpnlsvr" method="GET">
                      <div align="center">
                        <input type="hidden" name="object"
value="PROG">
                        <input type="hidden" name="operate"
value="RUN">
                        <input type="hidden" name="pname"
value="PART1">
                        <input type="submit" value="Run TP Program
Part1">
                      </div>
                    </form>
                  </td>
            </tr>
            <tr>
                  <td height="30">
                    <form action="../../Karel/mpnlsvr" method="GET">
                      <div align="center">
                        <input type="hidden" name="object"
value="DOUT">
                        <input type="hidden" name="operate" value="set">
                        <input type="hidden" name="index" value="1">
                        <input type="hidden" name="value" value="ON">
                        <input type="submit" value="Auto Mode
(DOUT[1])">
                      </div>
                    </form>
              </td>
            </tr>
            <tr>
                  <td height="30">
                    <form action="../../Karel/mpnlsvr" method="GET">
                      <div align="center">
                        <input type="hidden" name="object"
```

```
    value="sysvar">
                                    <input type="hidden" name="operate"
    value="setint">
                                    <input type="hidden" name="vname"
    value="$MCR.$GENOVERRIDE">
                                    <input type="hidden" name="value" value="100">
                                    <input type="submit" value="GenOverride=100">
                                  </div>
                                </form>
                              </td>
                </tr>
              </table>
            </div>
          </td>
        </tr>
    </table>
</div>

</body>

</html>
```

12.4.3 mplsvr.kl

```
-- Example karel program

program mpnlsvr

%NOLOCKGROUP
%NOABORT=ERROR+COMMAND
%NOPAUSE=ERROR+COMMAND+TPENABLE
%NOBUSYLAMP
%ENVIRONMENT REGOPE
%ENVIRONMENT SYSDEF
%ENVIRONMENT KCLOP

var

-- Declare HTML parameter names and value
   object   : string[12]
   pname    : string[12]
   operate : string[12]
   index    : string[12]
   value    : string[12]
   URL      : string[128]
   vname    : string[128]

-- These are duplicates that will be used to
-- convert the input parameters to Upper case
```

```
    uobject : string[12]
    uoperate: string[12]
    uindex    : string[12]
    uvalue    : string[12]
    upname    : string[12]
    uvname    : string[128]

-- Misc Variables

    kcommand : string[126]

    value_i : integer
    value_r : real
    index_i : integer
    status    : integer
    i         : integer
    index_p : integer
    entry     : integer

    file1     : file
    return_code : integer

-- Convert input string to Uppercase for consistant comparison

routine toupper(p_char: integer): string

begin
    if (p_char > 96) and (p_char < 123) then
       p_char = p_char - 32
    endif
    return (chr(p_char))
end toupper

begin

-- Good practice to check for uninitialized variables before using them

    if uninit(object)    then object    = ''; endif
    if uninit(operate) then operate = ''; endif
    if uninit(index)     then index     = ''; endif
    if uninit(value)     then value     = ''; endif
    if uninit(pname)     then pname     = ''; endif
    if uninit(vname)     then vname     = ''; endif

-- Change all character of input parameters to uppercase for string comparision

    uobject = ''
    for i = 1 to str_len(object) do
       uobject = uobject + toupper(ord(object, i))
    endfor
```

```
   uoperate = ''
   for i = 1 to str_len(operate) do
      uoperate = uoperate + toupper(ord(operate, i))
   endfor

   uindex = ''
   for i = 1 to str_len(index) do
      uindex = uindex + toupper(ord(index, i))
   endfor

   uvalue = ''
   for i = 1 to str_len(value) do
      uvalue = uvalue + toupper(ord(value, i))
   endfor

   upname = ''
   for i = 1 to str_len(pname) do
      upname = upname + toupper(ord(pname, i))
   endfor

   uvname = ''
   for i = 1 to str_len(vname) do
      uvname = uvname + toupper(ord(vname, i))
   endfor

-- Handle setting DOUTs

   if (uobject = 'DOUT') then
      if (uoperate = 'SET') then
         cnv_str_int(uindex, index_i)
         if (uvalue = 'ON') then
            DOUT[index_i] = ON
         endif
         if (uvalue = 'OFF') then
            DOUT[index_i] = OFF
         endif
      endif
   endif

-- Handle Setting OPOUTs

   if (uobject = 'OPOUT') then
      if (uoperate = 'SET') then
         cnv_str_int(uindex, index_i)
         if (uvalue = 'ON') then
            OPOUT[index_i] = ON
         endif
         if (uvalue = 'OFF') then
            OPOUT[index_i] = OFF
```

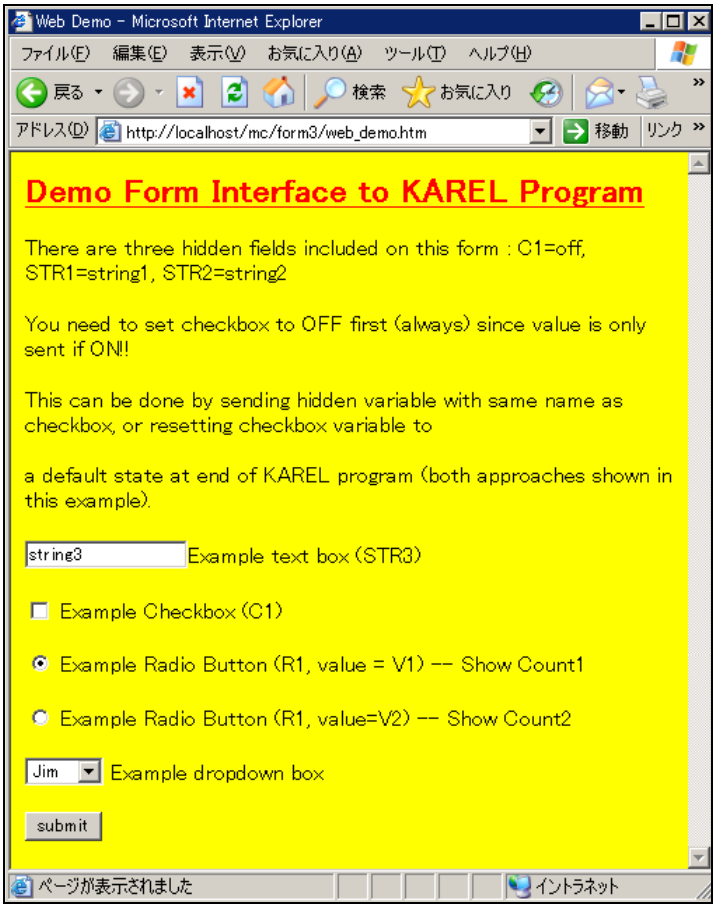
```
        endif
      endif
   endif

-- Handle Setting GOUTs

   if (uobject = 'GOUT') then
      if (uoperate = 'SET') then
         cnv_str_int(uindex, index_i)
         cnv_str_int(uvalue, value_i)
         GOUT[index_i] = value_i
      endif
   endif

-- Handle Setting Numreg values

   if (uobject = 'NUMREG') then
         cnv_str_int(uindex, index_i)
      if (uoperate = 'SETINT') then
         cnv_str_int(uvalue, value_i)
         set_int_reg(index_i, value_i, status)
      endif

      if (uoperate = 'SETREAL') then
         cnv_str_real(uvalue, value_r)
         set_real_reg(index_i, value_r, status)
      endif
   endif

-- Handle Running and Aborting a program

   if (uobject = 'PROG') then
      if (uoperate = 'RUN') then
    kcommand = 'RUN '+ upname
         KCL_no_wait (kcommand, status)
      else
    kcommand = 'ABORT ' + upname
    KCL_no_wait (kcommand, status)
      endif
   endif

-- Handle Setting a System Variable

   if (uobject = 'SYSVAR') then
      if (uoperate = 'SETINT') then
         cnv_str_int(uvalue, value_i)
         set_var(entry, '*SYSTEM*', uvname, value_i, status)
      endif

      if (uoperate = 'SETREAL') then
```

```
        cnv_str_real(uvalue, value_r)
        set_var(entry, '*SYSTEM*', uvname, value_r, status)
      endif

      if (uoperate = 'SETSTR') then
        set_var(entry, '*SYSTEM*', uvname, uvalue, status)
      endif
    endif

-- Return a NO RESPONSE Required code

    return_code = 204

end mpnlsvr
```

12.5 FORM USAGE SAMPLE 3

12.5.1 Overview

This subsection is almost same of Ethernet Function OPERATOR'S MANUAL (B-82974EN/01), 6.3.6 Creating Web Pages Based on KAREL Programs. Refer it to see detail. The sample of this subsection is following.



This page makes from two elements.

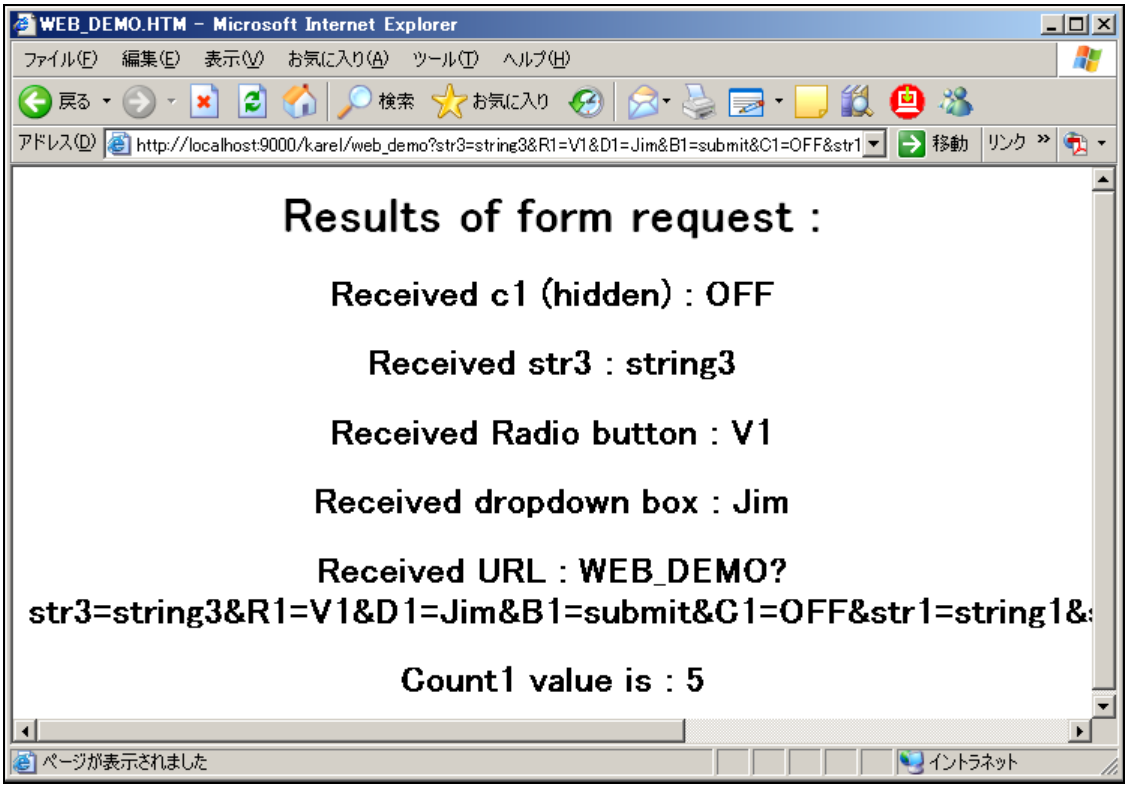
- /mc/form3/form.stm :Web page

- web_demo.kl :KAREL program

There are following functions.

- KAREL program, web_demo is started after pushing submit button.
- Then the value of text box or radio button is set to corresponding variables.
- Specified variables are also set as <input type="hidden" name="str1" value="string1">.
- Result is shown by web_demo depending on variable

In addition, the status of radio button in time of displaying pages does not reflect the content of variable.



12.5.2 web_demo.stm

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=shift_jis">
<title>Web Demo</title>
</head>
<body bgcolor="#FFFF00">
<p><font color="#FF0000" size="5"><strong><u>Demo Form Interface to KAREL
    Program</u></strong></font></p>
<form action="../../../karel/web_demo" method="GET" name="uif_demo">
<p>There are three hidden fields included on this form : C1=off, STR1=string1,
    STR2=string2</p>
<p>You need to set checkbox to OFF first (always) since value is only sent if
    ON!!</p>
<p>This can be done by sending hidden variable with same name as checkbox, or
    resetting checkbox variable to</p>
<p>a default state at end of KAREL program (both approaches shown in this
```

```
    example).</p>
<p><input type="text" size="20" name="str3"value="string3">Example text box
    (STR3)</p>
<p><input type="checkbox" name="C1" value="ON">
Example Checkbox (C1)</p>
<p>
<input type="radio" checked name="R1" value="V1">
Example Radio Button (R1, value = V1) -- Show Count1</p>
<p><input type="radio" name="R1" value="V2">
Example Radio Button (R1, value=V2) -- Show Count2</p>
<p><select name="D1" size="1">
<option>Jim</option>
<option>Joe</option>
<option>Harry</option>
</select> Example dropdown box</p>
<p><input type="submit" name="B1"
value="submit"></p>
<input type="hidden" name="C1" value="OFF">
<input type="hidden" name="str1" value="string1">
<input type="hidden" name="str2" value="string2">
</form>
</body>
</html>
```

12.5.3 web_demo.kl

```
-- Example KAREL program to respond to a form called web_demo.htm created in
-- frontpage. Note that form data are is populated in corresponding KAREL
-- variables, if Variables are declared. String variable called URL should
-- be declared to see exactly what is provided from browser which is useful
-- for debugging.---- Example of received UR+ :
--
    WEB_DEMO?STR1=STRING1&STR2=STRING2&STR3=STRING3&C1=ON&R
    1=V1&D1=JIM&B1=SUBMIT
---- NOTE: variables which are included in URL are populated each time
-- the program is called. Some form variables (eg. checkbox) are only
-- sent if they are checked. This behavior can be handled by always
-- passing a "hidden" variable of same name with default value from
-- form, or by resetting variables with this nature to a default state
-- after program runs (see c1 variable assignment at end of this program).
-- Program variables are uninitialized the first time a program runs
-- (aside from ones which are set by URL, since any variables included in
-- URL are set before program is called).
--PROGRAM web_demo
%NOLOCKGROUP
CONST
    TEXTHDR = '<HTML> <BODY>'
    TEXTTRLR = '</BODY> </HTML>'
    BACKGROUND = 'FRS/EARTHBG.GIF' -- used in MD_FILES.HTM
```

```
    PIC1 = 'FR/PICTURE.GIF' -- some picture for top of response file
VAR
    count1 : integer
    count2 : integer
    file1 : FILE
    URL : STRING[128]
    str1 : STRING[12]
    str2 : STRING[12]
    str3 : STRING[12]
    c1 : STRING[12]
    r1 : STRING[12]
    d1 : STRING[12]
BEGIN
    -- Good practice to check for uninitilialized variables before using
    -- them
    IF UNINIT(count1) THEN count1 = 0; ENDIF
    IF UNINIT(str1) THEN str1 = ''; ENDIF
    IF UNINIT(str2) THEN str2 = ''; ENDIF
    IF UNINIT(str3) THEN str3 = ''; ENDIF
    IF UNINIT(c1) THEN c1 = ''; ENDIF
    IF UNINIT(r1) THEN r1 = ''; ENDIF
    IF UNINIT(d1) THEN d1 = ''; ENDIF
    IF UNINIT(URL) THEN url = ''; ENDIF
    count1 = count1 + 1 -- these might be production counts from another program
    count2 = count1 * 2 -- they are just included as examples
OPEN FILE file1 ('RW', 'RD:RESPONSE.HTM')
WRITE file1('<HTML><HEAD><TITLE>WEB_DEMO.HTM</TITLE></HEAD>',CR)
WRITE file1('<BODY BACKGROUND="../')
WRITE file1(BACKGROUND)
WRITE file1('">',CR)
-- Could add a graphic to top of response file
-- write file1('<CENTER> <H1><A NAME="TOP"><IMG SRC="../')
-- write file1(PIC1,cr)
-- write file1('" WIDTH="400" HEIGHT="100"></A></H1> </CENTER>',cr)
-- write file1('"></A></H1> </CENTER>',cr)
WRITE file1('<H1><CENTER><BOLD>Results of form request :',CR,CR)
WRITE file1('</CENTER></H1>',CR)
-- checkbox only sent if check so send default state always
WRITE file1('<H2><CENTER><BOLD>Received c1 (hidden) : ')
WRITE file1(c1,CR)    WRITE file1('</BOLD></CENTER></H2>',CR)
WRITE file1('<H2><CENTER><BOLD>Received str3 : ')
WRITE file1(str3,CR)
WRITE file1('</BOLD></CENTER></H2>',CR)
IF (c1='ON') THEN
WRITE file1('<H2><CENTER><BOLD>Received Checkbox : ')
WRITE file1(c1,CR)
WRITE file1('</BOLD></CENTER></H2>',CR)
ENDIF
WRITE file1('<H2><CENTER><BOLD>Received Radio button : ')
WRITE file1(r1,CR)
```

```
WRITE file1('</BOLD></CENTER></H2>',CR)
WRITE file1('<H2><CENTER><BOLD>Received dropdown box : ')
WRITE file1(d1,CR)
WRITE file1('</BOLD></CENTER></H2>',CR)
WRITE file1('<H2><CENTER><BOLD>Received URL : ')
WRITE file1(URL,CR)
WRITE file1('</BOLD></CENTER></H2>',CR)
IF (r1='V1') THEN
    WRITE file1('<H2><CENTER><BOLD>Count1 value is : ')
    WRITE file1(count1,CR)
    WRITE file1('</BOLD></CENTER></H2>',CR)
else
    WRITE file1('<H2><CENTER><BOLD>Count2 value is : ')
    WRITE file1(count2,CR)
    WRITE file1('</BOLD></CENTER></H2>',CR)
ENDIF
-- If default value of checkbox is not sent as hidden variable, another
-- alternative is to rest checkbox variable to default state after
-- program runs. As with all KAREL programs, global variables retain
-- their value between each execution
c1 = 'OFF'
    WRITE file1(TEXTTRLR,CR)
    CLOSE FILE file1
END web_demo
```

13 NOTE OF USING KAREL

- In R-30*i*A robot controller, created PC files of KAREL, VR files including variables are allowed to backup from FILE screen by all backup operation. However VR files made from ftx files for Form are not backuped before 7DA4 series (including 7DA4). Also it cannot load all from controlled start (it can load from FILE screen individually). No VR file for Form is not loaded with all load operation because they are not saved by all backup operation. Load VR files created from ROBOGUIDE after all load operation.

- Form Editor cannot use in the robot controller until 7DA1 series (including 7DA1). Use newer than 7DA3 series to use Form Editor.

- Be careful to use global variables in case that they are accessed in multi process. They are overwritten in unexpected part for KAREL program create and unexpected process may be executed. To create a KAREL program not to refer global variables from multi part may be do debug effectively.

- WRITE statement gets KAREL program processing speed lower. You must remove WRITE statement before product line is run after debugging completely.

- Error is posted when it refers to uninitilized variables. Some built-in functions allow using uninitilized variables. But you should initialize variables.

INDEX

Revision Record

FANUC R-30*i*A/R-30*i*A Mate CONTROLLER KAREL Function
OPERATOR'S MANUAL (B-83144EN)

Edition	Date	Contents	Edition	Date	Contents
01	Mar.,2010	———			

B-83144EN/01

* B - 8 3 1 4 4 E N / 0 1 *